

64'er
Das Sonderheft für C128-Fans
Sonderheft Nr. 0001
ÖS 100,-/sfr. 14,-
Lit. 12 000/hfl. 18,-/dkr. 68,-
DM 14,-
Markt&Technik
128er
Das bringt der C128
★ Die Vorteile des Basic 7.0
★ CP/M – Das Profi-Betriebssystem
★ Fantastische Grafik einfach programmiert
Die Unterschiede zum C64
Das paßt zum C128
★ 3 Floppies im Test
★ Die besten Monitore
★ Drucker für jeden Zweck
Tolle Tips, Tricks und Listings
Markt&Technik
64'er
DAS MAGAZIN FÜR COMPUTER-FANS
Alle Programme auch auf Diskette erhältlich



Alles über den Commodore 128

Endlich eine komplette Beschreibung des Gesamtsystems. Was taugt das neue Basic 7.0? Wie kompatibel ist der C128 zum C64? Was ist CP/M? Was gibt es an Software? Diese und viele weitere Fragen werden ausführlich beantwortet, um jeden am C128 Interessierten über die Fähigkeiten dieses Computers zu informieren und eine echte Entscheidungshilfe zu geben.



Alles über die C128-Hardware und Software

Natürlich werden auch die neuen Diskettenlaufwerke 1570 und 1571 vorgestellt und auf Herz und Nieren getestet. Wir sagen Ihnen, welches Laufwerk für welche Anwendung geeignet ist und wo die Vorteile der neuen Commodore-Laufwerke liegen. Auch das leidige Monitor-Thema wird angesprochen. Unsere Selbstbauanleitung für eine 40/80-Zeichen-Umschaltplatine hilft bares Geld sparen und erhöht den Bedienungskomfort ganz erheblich.

Auch bei der Vorstellung der Hardware gab es keine Kompromisse: Fakten statt Fiktionen, das ist das zentrale Motto, unter dem dieses Sonderheft steht. Auf über zwanzig Seiten erfahren Sie alles über die Hardware des C128 und wie sie über die Software gesteuert wird. Begriffe wie MMU, Bank Switching, VDC-Controler oder DMA werden erklärt und das Zusammenwirken der Hard- und Software-Komponenten des Betriebssystems wird entschleiert.

Durch die 80-Zeichen-Option und das CP/M-Betriebssystem ist der C128 im Gegensatz zum C64, dessen Software er auch ausführen kann, nicht nur für den Heimbereich, sondern auch für professionellere Anwendungen geeignet. So ist es für uns selbstverständlich, Ihnen mit diesem Sonderheft auch einen Überblick über bereits erhältliche professionelle Anwendersoftware unter CP/M zu geben. Sie finden daher Test- und Anwendungsberichte über das Textverarbeitungsprogramm Wordstar, das Datenbanksystem dBase II sowie für das Tabellenkalkulationsprogramm Multiplan. Der Testbericht über Turbo-Pascal stellt Ihnen den leistungsfähigsten und schnellsten Pascal-Compiler im 8-Bit-Bereich vor.

Für den Basic-Programmierer gibt es eine leichtverständliche Einführung in das Arbeiten mit dem 7.0-Basic, insbesondere was das interessante Gebiet der strukturierten Programmierung angeht. Sehen Sie selbst anhand von kleinen Beispielprogrammen, wieviel komfortabler das C128-Basic speziell gegenüber dem doch eher spartanischen V2.0-Basic des C64 ist.

Für passionierte Assembler-Programmierer bringen wir interessante Informationen über das Betriebssystem, insbesondere über das Speicher-Management des C128. Klipp und klar wird endlich einmal gesagt, welche Speicherbereiche Sie für Assembler-Routinen benutzen können, was ein »FAR JSR« ist und wie man ihn benutzt, wie Sie ROM-Routinen nutzen und Basic-Erweiterungen schreiben können.

Natürlich finden Sie in diesem Sonderheft auch eine Menge interessanter Tips und Tricks-Listings: Doppelte Grafikauflösung von 640 x 200 Punkten per Basic-Kommandos (ein Maschinenprogramm, das auf jede C128-Utility-Diskette gehört), Cursor-Manipulationen unter CP/M und vieles andere mehr.

Doch genug der einleitenden Worte. Wenn Sie, unsere Leser, uns weiterhin die Treue halten und mit Rat und Tat und Listings unterstützen, dann versprechen wir Ihnen, daß Sie nicht zum letztenmal ein ganzes Sonderheft randvoll mit Informationen, Tips und Tricks zum C128, eben ein »128er«, in den Händen halten. (ev)



Bestellungen aus anderen Ländern bitte per Auslandspostanweisung!

Bestellungen aus der Schweiz bitte direkt an:
Markt&Technik Vertriebs AG
Kollerstrasse 3
CH-6300 Zug
Tel. 042/41 56 56

Bestellungen aus Österreich bitte direkt an:
Bücherzentrum Meidling
Schönbrunnerstr. 261
1120 Wien
Tel. 0222/8331 96
Microcomput-ique
Erhard Schiller
Fasangasse 21
1030 Wien
Tel. 0222/785661

Hardware für alle - ein neuer 64'er Leser-Service

Der Commodore 64 hat schon oft bewiesen, wie vielseitig er ist. Er läßt sich nicht nur mit Programmen, sondern auch durch so manche Hardware-Erweiterung sinnvoll nutzen und ausbauen. Dabei ist es sicherlich ein reizvoller Bestandteil des Computer-Hobbys, sich solche Erweiterungen selbst nachzubauen. Aber nicht jeder Leser verfügt über die Gelegenheit und Zeit zur Platinenherstellung. Hinzu kommt, daß es oft zu teuer ist, wegen einer bestimmten Erweiterung, Investitionen von mehreren hundert Mark für eine Platinenstation zu tätigen. Wir haben reagiert: Ab sofort besteht die Möglichkeit, im Rahmen des Leser-Service, die in der 64'er abgedruckten Hardware-Erweiterungen in drei verschiedenen Ausbaustufen zu erhalten:

1. Als Platinen
Nur Leerplatinen. Die Beschaffung der Bauteile und der Zusammenbau bleibt bei Ihnen.

2. Als Bausätze
Unsere Bausätze enthalten alle Teile, die notwendig sind, um die beschriebene Erweiterung komplett aufzubauen. Sie brauchen die Bauteile nur noch gemäß der Anleitung in dem jeweiligen Heft zusammenzulöten und einzubauen.

3. Als Fertiggeräte
Die Fertiggeräte sind komplett aufgebaute und geprüfte Geräte. Sie brauchen die Erweiterung lediglich noch einzubauen.

Wichtiger Hinweis: Wir bemühen uns um eine umgehende Auslieferung Ihrer bestellten Hardware. Aber bis zum Eingang Ihrer Überweisung, der Auftragsabwicklung und der dazugehörenden Postwege vergehen mindestens 3 Wochen. Bitte haben Sie Verständnis, wenn aus diesen Gründen Ihre Hardware nicht sofort bei Ihnen eintrifft.

Unser Angebot

Angebot 1:

Expansion-Port Eprom-Platine mit 1 x 8 KByte Speicherplatz für 2732 bis 2764 Eproms.

Beschreibung in Ausgabe 10/85
Bestellnummer: HW 010 — pro Stück **19,80***
Dieser Artikel wird nur als Fertiggerät angeboten.

Angebot 2:

Expansion-Port Eprom-Platine mit 2 x 8 KByte Speicherplatz für 2732 bis 2764 Eproms, mit Umschaltmöglichkeit.

Beschreibung in Ausgabe 10/85

Leerplatine
Bestellnummer: HW 020 — pro Stück **24,80***

Bausatz mit allen Teilen:
Bestellnummer: HW 021 — pro Stück **49,80***

Fertiggerät, getestet, wie beschrieben:
Bestellnummer: HW 022 — pro Stück **59,80***

Angebot 3:

Eprom Trans – Die Speichererweiterung

ROM-Speichererweiterung zum Einbau in den C64, gleichzeitig Steckplatz für ein Original- oder ein alternatives Betriebssystem. Zwei Platinen in Epoxid-Harz-Ausführung wie in Ausgabe 10/85 beschrieben.

Leerplatine
Bestellnummer: HW 030 — pro Stück **49,80***

Bausatz mit allen Teilen:
Bestellnummer: HW 031 — pro Stück **119,80***

Eprom-Trans ist nicht als Fertiggerät erhältlich. Die Hardware-Erweiterungen aus früheren Ausgaben und die 40/80 Zeichen-Umschaltung für den C128 werden wir so bald als möglich in unser Angebot aufnehmen.

Angebot 4:

Super Kernal

Erweitertes Betriebssystem für den C 64 mit vielen neuen Funktionen inkl. Adaptersockel, einbaufertig in den C 64.
Beschreibung in Ausgabe 11/85

Version 1: Enthält Hypra Load / DOS 5.1 / Funktionstastenbelegung / Renew / RS232
Bestellnummer: HW 040

Version 2: Enthält Hypra Load / DOS 5.1 / Funktionstastenbelegung / Renew / Super Centronics Schnittstelle
Bestellnummer: HW 041

Version 3: Enthält Hypra Load / DOS 5.1 / Funktionstastenbelegung / Renew / Hypra Save
Bestellnummer: HW 042

Version 4: Enthält Hypra Load / DOS 5.1 / Funktionstasten / Hypra Save / Centronics klein
Bestellnummer: HW 043

Preis für jede Version pro Stück: **39,80***

* Alle Preise inklusive Mehrwertsteuer

Qualität & Service

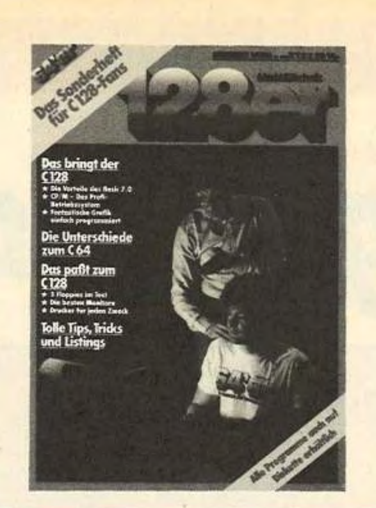
- Die 64'er Hardware hat einen hohen Qualitätsstandard. Wir verwenden nur beste Epoxid-Harz-Platinen mit Lötstopp-Lack.
- Wir verwenden nur Präzisionssockel mit gedrehten Kontakten.
- Alle Platinen werden professionell gefertigt. Wenn notwendig mit doppelseitiger Beschichtung und Löt-Durchkontaktierungen.
- Jedes Gerät, das wir versenden, wurde auf Funktionstüchtigkeit geprüft.
- Wir sind auch nach dem Verkauf für Sie da.
  Neben der gesetzlichen Garantie bietet unser Service- und Fertigungspartner Ihnen Hilfe und Unterstützung an.

Unsere Garantie

Im Rahmen der Versand- und Lieferbedingungen unterliegen die Geräte einer Gewährleistungszeit von 6 Monaten ab Lieferung. Der Lieferung liegt eine Service-Karte bei, die Sie im Falle einer Beanstandung zusammen mit dem Gerät an die auf der Karte vermerkte Adresse schicken können. Die gleiche Karte verwenden Sie bitte bei Reparaturen nach der Garantiezeit.

Wie bestelle ich?

Alle Hardware-Erweiterungen, die Sie bestellen können, tragen einen Bestellverweis am Ende des Artikels im jeweiligen Heft. Falls Sie keinen Hinweis finden, hat sich der Autor dieser Erweiterung nicht dazu entschließen können, seine Entwicklung im Rahmen des Leserservice für eine Verbreitung freizugeben. Bitte verwenden Sie für Ihre Bestellung immer die beiliegende Postscheck-Zahlkarte oder einen Verrechnungsscheck. Sie erleichtern uns damit die Auftragsabwicklung und sparen sich Versandkosten.


Das Sonderheft für C128-Fans
128er
Das bringt der C128
Die Unterschiede zum C64
Das paßt zum C128
Tolle Tips, Tricks und Listings

INHALT


128er

Rundgang durch die

Wir zeigen Ihnen, was alles im C128 steckt. In einem »Rundgang« über die Platine des C128 erklären wir die Funktionen der einzelnen Bauteile und was man damit anfangen kann.

Seit etwa einem halben Jahr gibt es ihn schon, den neuesten aus dem Hause Commodore, den C128. Einerseits wartet der C128 mit so mancher angenehmen Überraschung auf, wie etwa 128 KByte RAM, dem CP/M-Modus und der Kompatibilität zum C64. Andererseits läßt aber auch er nicht die von Commodore, und auch anderen Computerherstellern gewohnten »Kompromisse« vermissen. Zu nennen wäre da beispielsweise die Tatsache, daß man einen Monitor braucht der sowohl RGB- als auch compositefähig ist, um sämtliche Grafikmöglichkeiten auszunützen, die im C128 stecken. Bei einem Blick auf die Platine des C128 trifft man manchen vom C64 her bekannten Baustein. Den Sound-Chip SID 6581 etwa, oder den Complex-Interface-Adapter CIA 6526. Bild 1 zeigt die »nackte« Platine. Sämtliches »Drumherum« wie Gehäuse, Tastatur etc. wurde entfernt. Man erkennt deutlich, daß es auf der Platine des C128 viel gedrängter zugeht als auf der Platine des C64, zu dem der C128 ja 100%ig kompatibel sein soll. Bei dieser Aussage kam natürlich Skepsis in uns auf und wir testeten eine Vielzahl von verschiedenen Programmen, die intensiven Gebrauch von den Eigenheiten des C64 machen (Illegal Opcodes und Betriebssystem-Routinen). Das Ergebnis war überraschend, denn der C128 zeigte sich im C64-Modus fast vollkommen kompatibel zu seinem Vorgänger.

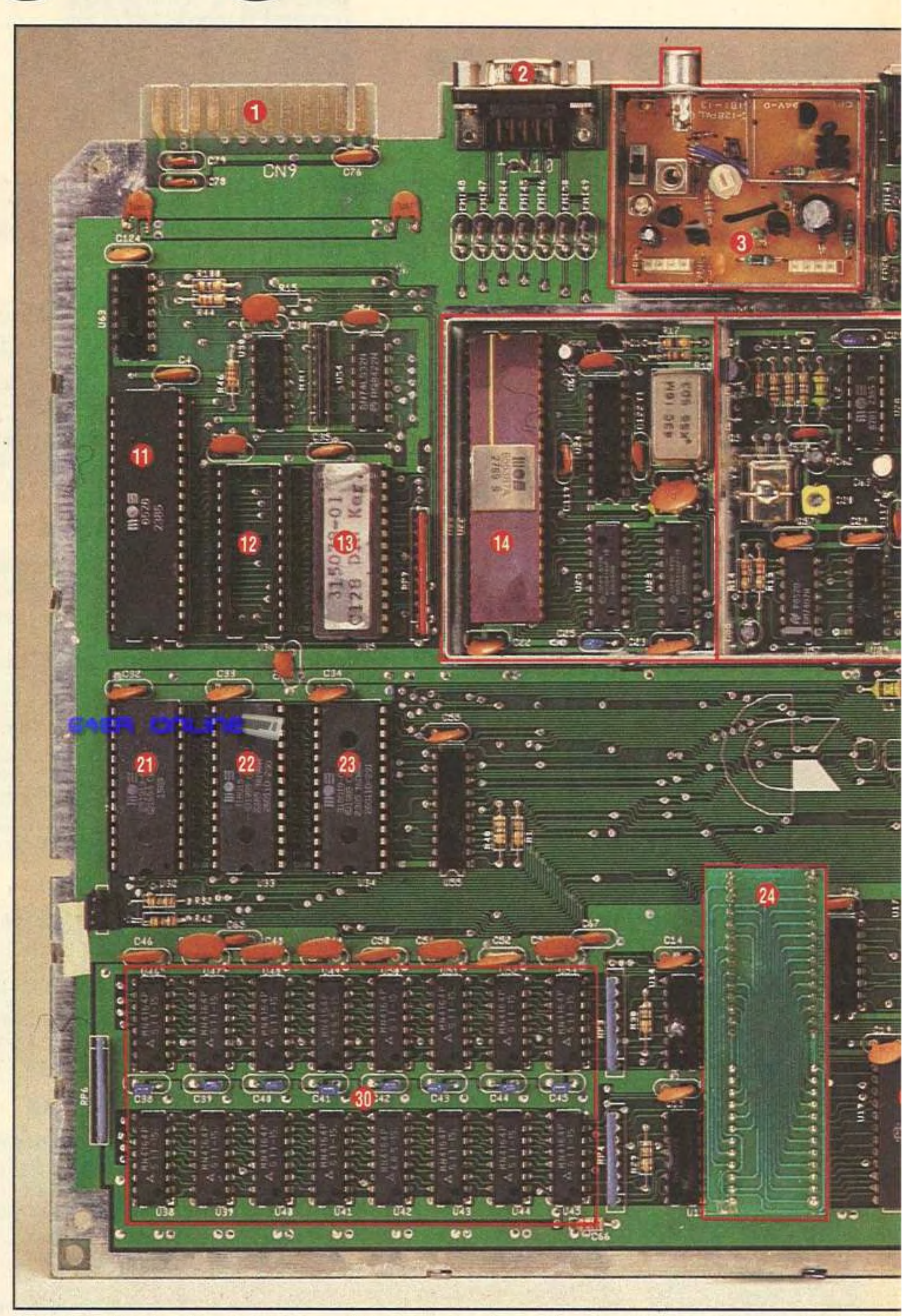


Bild 1. Die Platine des C128. Unsere Wanderung beginnt rechts oben beim Anschluß des Netzteils.

Der C128 bietet eine ganze Menge verschiedener Betriebsmodi. Nach dem Einschalten liegt die eigentliche C128-Betriebsart vor, in der sich der neue Basic-Interpreter 7.0 von seiner besten Seite zeigt. Weitere Modi sind der bereits erwähnte C64-Modus und der CP/M-Modus. Im CP/M-Modus verhält sich der C128 wie eine echte CP/M-Maschine, ist aber bedingt durch die geringere Taktfrequenz nicht ganz so schnell. Für den CP/M-Modus erhielt der C128 neben dem Prozessor 8502 (kompatibel zur 65xx-Familie) einen Z80 B-Prozessor.

Neben den drei verschiedenen Modi kann der Benutzer, außer im CP/M-Modus, die Taktfrequenz auf 1 MHz oder 2 MHz festlegen, allerdings mit Einschränkungen.

Bekannte Hardware

In Bild 1 sind die einzelnen Bauelemente und Baugruppen bezeichnet. Wir wollen nun einen kleinen Rundgang über die Platine des C128 machen und uns dabei die Aufgaben und Funktionsweisen der verschiedenen Bauteile verdeutlichen. Unser Wanderweg startet rechts oben am Netzteilan-



Hardware des C 128

1 User-Port
2 Der RGB-Ausgang ist anschlußkompatibel zu IBM-Farb- und Schwarzweiß-Monitoren. Die 80-Zeichendarstellung geht nur über diesen Ausgang.
3 HF-Modulator. Hier wird ein Hochfrequenz-Bildsignal erzeugt, das ein normaler Fernseher »versteht«.
4 Der Video-Audio-Port entspricht dem des C 64. Er dient zur 40-Zeichen- und Grafikdarstellung.
5 Der serielle Port des C 128 entspricht dem des C 64. Im C 128-Modus arbeitet er aber etwa 8mal schneller.
6 Kassetten-Port.
7 Expansion-Port für den C 64- und C 128-Modus.
8 Netzteilanschluß.
9 Ein-/Ausschalter.
10 Reset-Taster.
11 CIA 2. I/O-Baustein für den User-Port und den seriellen Bus.
12 32 KByte ROM. Frei für eigene ROMs.
13 16 KByte ROM. C 128, Betriebssystem (Kernel) und 40/80-Zeicheneditor.
14 RGB-Teil. Links der 80-Zeichen-Videocontroller 8563 (RGB). Rechts unten, das 16-KByte/8-Bit-Video-RAM für den 8563.
15 Composite-Video-Teil. Rechts der VIC 8564, der dem VIC aus dem C 64 beinahe identisch ist. Er enthält einen eigenen Grafikprozessor.
16 Der inzwischen schon altbekannte SID, der 6581 aus dem C 64.
17 Die MMU bestimmt aus welchem Speicherteil der nächste Zugriff stattfinden soll.
18 Entprell-Kondensatoren für Control-Ports.
19 Control-Port 2.
20 Control-Port 1.
21 Betriebssystem- und Basic-ROM für den C 64-Modus. 16 KByte ROM.
22 16 KByte ROM. C 128, Basic Teil 2.
23 16 KByte ROM. C 128, Basic Teil 1.
24 Unter dieser »abenteuerlichen« Konstruktion befindet sich der deutsche und der ASCII-Zeichensatz.
25 Die PLA, ein 8721. Die PLA ist ein Gatterbaustein mit programmierbaren Logikfunktionen. Sie bestimmt, neben der MMU, welche ROMs aktiviert werden.
26 Der 8502-Prozessor.
27 Die Z80B-CPU.
28 CIA 1. Hier werden die gedrückten Tasten festgestellt und die Control-Ports abgefragt.
29 Entprell-Kondensatoren der Tastatur.
30 128 KByte RAM. 16 8-KByte/8-Bit-RAMs.
31 2-KByte/8-Bit-CMOS-RAM. Der Farbspeicher für den VIC 8564.
32 Tastatur-Steckerliste.

schluß, ohne den ja nichts »laufen« würde. Ohne elektrischen Strom, sprich Elektronen, kann bekanntlich kein Computer funktionieren. Wir marschieren also weiter in Richtung des unteren Platinenrandes und kommen dabei am Ein/Aus-Schalter vorbei, dessen Funktion wohl auch bekannt sein dürfte. Lassen wir ihn also rechts liegen und gehen weiter in Richtung Platinenrand. Gleich neben dem Ein/Aus-Schalter liegt der Reset-Taster, der bei Commodore bis vor kurzem keineswegs Standard war. Ein Druck auf diesen Taster setzt den Computer in den Einschaltzustand zurück. Besonders sinnvoll ist der Reset-Taster für Assembler-Programmierer. Es kommt nur allzuoft vor, daß ein Maschinenprogramm nicht auf Anhieb so funktioniert, wie es eigentlich geplant war. Meist endet ein solches Maschinenprogramm in einer endlosen Schleife, aus der der Computer dann mit dem Reset-Taster »zurückgeholt« werden kann.

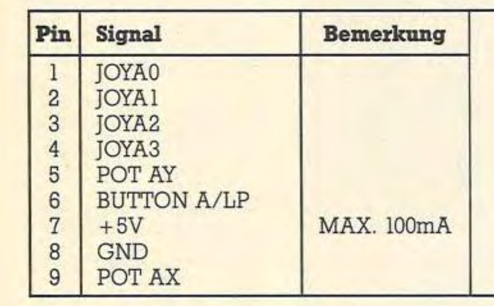
Doch gehen wir weiter auf den unteren Rand der Platine zu, und bleiben etwa zwischen den beiden Control-Ports (Bild 2 und Bild 3) stehen. Ein Blick zwischen den Ent-

prellkondensatoren hindurch zeigt am rechten Rand die zwei Control-Ports. Diese Control-Ports sind mit denen des C64 vollkommen identisch. Es können hier zwei Joysticks oder vier Paddles angeschlossen werden. Lightpen und Trackball werden ebenfalls über diese Schnittstelle mit dem Computer verbunden. Die Steuerung der Control-Ports erfolgt über CIA 1, an der wir noch vorbeikommen werden. Die vielen Kondensatoren bis hinunter zur Tastatur-Anschlußleiste haben die Aufgabe, die Kontakte des Joysticks und der Tastatur elektrisch zu »entprellen«. Wird ein mechanischer Schalter geschlossen, ist im Gegensatz zu elektronischen Schaltern, wie zum Beispiel einem Transistor, zu beobachten, daß der Kontakt, bevor er endgültig zustandekommt, einige Male unterbrochen wird.

Der Elektroniker nennt dieses Phänomen »Kontaktprellen«. Bild 4 zeigt, wie die Entprellkondensatoren den Schaltimpuls »glätten«.

Commodores Entwickler haben deshalb zu einem Trick gegriffen, um 128 KByte RAM adressieren zu können, ohne einen leistungsfähigeren Prozessor einzusetzen. Die Lösung heißt »Bank-Switching«. Der Speicher von 128 KByte wird in zwei Speicher-Bänke von je 64 KByte unterteilt. Der Prozessor greift dann wechselweise auf die eine oder andere Speicherbank zu. Hier ist auch die Hauptaufgabe der MMU zu finden. Sie sorgt dafür, daß der Prozessor stets die richtige Speicherbank vorfindet.

Wer allerdings glaubt, nun endgültig alle Speicherprobleme gelöst zu sehen, dem sei gesagt, daß, wie auch schon beim C64, nicht der gesamte Speicherbereich für Basic-Programme zur Verfügung steht. Die zweite Bank (Bank 1) ist nämlich ausschließlich Basic-Variablen vorbehalten. Für die Programme bleibt also nur die erste Speicherbank frei (Bank 0) und von dieser zweigt sich das Betriebssystem noch einmal etwa 4 KByte für



Pin	Signal	Bemerkung
1	JOYA0	
2	JOYA1	
3	JOYA2	
4	JOYA3	
5	POT AY	
6	BUTTON A/LP	
7	+5V	MAX. 100mA
8	GND	
9	POT AX	



Bild 2. Die Anschlußbelegung der Control-Ports

Ohne die Entprellung könnte es unter Umständen dazu kommen, daß der Computer das Nachprellen des Kontaktes als weitere Tastendrücke interpretiert und so bei einem einmaligen Druck auf eine Taste gleich mehrmals das gewünschte Zeichen anzeigen würde.

Wenden wir unsere Aufmerksamkeit aber einmal nach links. Dort sehen wir in ihrem relativ großen Gehäuse einen sehr wichtigen Baustein des C128, die »Memory-Management-Unit« oder kurz MMU (Tabelle 1). Memory-Management-Unit bedeutet Speicherverwaltungs-Einheit, woraus auch schon die Funktion dieses eigens für den C128 entwickelten Chips hervorgeht. Der C128 verfügt über 128KByte RAM. Sie werden sich sicherlich gefragt haben, wie der 8502 und der Z80 (Tabelle 2) überhaupt auf einen solch großen Speicherraum zugreifen können. Beide sind ja bekanntlich wegen ihrer 16 Adreßleitungen nur zur Adressierung von 2^{16} = 65536 Byte fähig.

Zero-Page, Prozessorstack und Bildschirmspeicher etc. ab.

Zero-Page und Stack erscheinen natürlich auch in Bank 1, denn der Prozessor kann ohne seinen Stack nicht auskommen. Es ist auch nicht möglich, den Basic-Speicher auf Kosten des Variablen-Speichers zu erweitern. Allerdings kann der Hauptspeicher noch um zwei Bänke von je 64 KByte RAM erweitert werden, was dann einen Speicher von 256 KByte ergibt. Der Benutzer kann sich Speicherblöcke von 1 bis 16 KByte definieren und diese an den Speicheranfang, das Speicherende oder an beide Orte gleichzeitig legen. Der Inhalt dieser Bereiche ist dann auf beiden Banken der gleiche. Das dürfte vor allem von Vorteil sein, wenn zwei Programme, auf verschiedenen Bänken, die gleichen Daten benutzen sollen. Zur Definition dieser Bereiche hat die MMU das RAM Configuration Register (RCR). Neben der Verwaltung des Schreib/Lese-Speichers ist die MMU aber auch noch für den ROM-Bereich zuständig. Sie sagt dem Prozessor, ob er seine Daten aus einem internen Kernel-ROM holen soll, oder aus einem externen ROM oder EPROM auf einem Steckmodul.

Speicher-Verwaltungsstelle: Die Configuration Register

Die interessantesten Register der MMU sind das schon erwähnte RAM Configuration Register und das Configuration Register (CR). Das CR kontrolliert die ROM-, RAM- und die I/O-Konfiguration des C128. Das Register hat die Adresse $D500 im I/O- und $FF00 im Kernel-Bereich. Das CR bei $D500 wird nur bei I/O-Zugriffen benötigt. Die MMU stellt sich dann das Register selbst ein. Findet kein I/O-Zugriff statt, ist das CR, mit den gesamten I/O-Routinen, in der Memory-Map nicht vorhanden, im Gegensatz zum CR bei $FF00, das ständig in der Memory-Map präsent ist.

Bit 0 des Configuration Registers (CR) regelt im C128-Modus den Speichertyp zwischen den Adressen $8000 und $BFFF. Sind beide Bits »0«, wird auf den oberen Basic-Teil in der Memory-Map (Bild 5), also den zweiten Basic-ROM-Teil zugegriffen.

Ist Bit 2 »1«, wird ein internes ROM eingeblendet, das ROM des deutschen Zeichensatzes. Die Tastatur besitzt dann die DIN-Belegung. Richtig interessant wird es erst, wenn nur Bit 3 »1« ist, dann wird nämlich im Bereich von $8000 bis $BFFF ein Steckmodul eingeblendet. Sind beide Bits »1« sieht der C128 in diesem Bereich nur RAM.

Die nächsten beiden Bits, 4 und 5, haben die gleiche Funktion wie Bit 2 und 3, nur bestimmen sie den Speicheraufbau im Bereich von $C000 bis $FFFF. Zu bemerken ist, daß der Speicherbereich von $D000 bis $DFFF ein »ROM-Loch« darstellt. Man kann die Bits 4 und 5 setzen wie man will, der Computer entscheidet, ob I/O-Bereich oder das Zeichensatz-ROM in diesem Bereich eingeblendet wird. Es ist also bei einem Steckmodul zu berücksichtigen, daß der Bereich von $D000 bis $DFFF tabu ist. Es können bis zu 32 KByte ROM eingeblendet werden, doppelt so viel wie beim C64. Bit 6 und 7 schließlich selektieren die RAM-Bank. Für die 128 KByte-Version des C128 ist nur Bit 6 wichtig. Ist es »0«, so ist Bank 0 ausgewählt, ist es »1«, Bank 1.

Welche CPU und welche Betriebsart (C64, C128 oder CP/M) eingeschaltet ist, kann mit dem



»Mode Configuration Register« eingestellt werden (Tabelle 1). Mit den »Page-Pointer-Registern« wurde im C 128 ein Verlagern von Zero-Page und Stack (normalerweise von $0000 bis $00FF beziehungsweise $0100 bis $01FF) in beliebige andere Speicherseiten ermöglicht. Dadurch wächst der Speicherbereich mit der schnellen Zero-Page-Adressierung und auch der Datenbereich des Stacks beträchtlich an. Eine Neubesetzung der Page-Pointer muß immer zuerst am höherwertigen Byte erfolgen. Zu guter Letzt sei noch das Version Register beschrieben, das einzige Register, aus dem für die Betriebssoftware die Größe des verfügbaren Speichers ersichtlich ist. Außerdem ist hier auch noch die Versionsnummer der MMU untergebracht. Wie aus dieser Beschreibung hervorgeht, ist die MMU also keinesfalls ein toter Baustein.

nutzt. Der Ein-/Ausgang des Schieberegisters ist am User-Port verfügbar. Mit diesem Schieberegister können zum Beispiel ein oder mehrere C 128 miteinander gekoppelt werden. Das Schieberegister dient dann zur Informationsübertragung. Die Echtzeituhr der CIA 6526 besitzt eine erhebliche Langzeitgenauigkeit, weil sie direkt mit der Netzfrequenz (50 Hz) getriggert wird. Die Handhabung der Echtzeituhr ist nicht ganz einfach, aber nach einiger Übung wird man das Vorhandensein dieser Uhr nicht mehr missen wollen. In der CIA sind 16 Steuerregister integriert. Um eines der Register anzusprechen muß einfach die Registernummer zur Basisadresse (CIA 1 = $DC00) addiert werden. Die sich durch diese Addition ergebende Adresse ist wie jede andere Speicherstelle zu behandeln. Die beiden Ports dieser CIA werden zur zu programmieren (zur Auswahl der CPU siehe unter MMU). Ein Vorteil des C 128 gegenüber dem C 64 liegt in der höheren Taktfrequenz der CPU. Der 8502 kann wahlweise mit 1 MHz oder 2 MHz getriggert werden, woraus ein Geschwindigkeitsvorteil von 100% gegenüber dem C 64 resultiert. Im C 64-Modus ist es ebenfalls möglich, durch »POKE 53296,1« die Taktfrequenz auf 2 MHz heraufzusetzen, wobei dann allerdings der Bildschirm verrückt spielt. Durch »POKE 53296,0« wird die Taktfrequenz auf 1 MHz zurückgesetzt. Der Z 80 wird sogar mit 4 MHz getaktet, allerdings nur, wenn er interne Operationen ausführt wie zum Beispiel arithmetische Operationen. Greift er dagegen auf das Bussystem zu, so wird von einer entsprechenden Logik die Taktfrequenz auf 2 MHz reduziert. Der Z 80 ist ja hinlänglich bekannt und millionen-

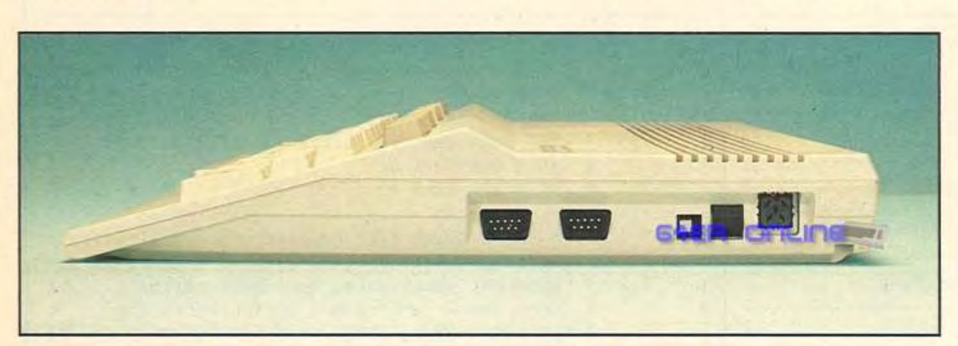


Bild 3. Seitenansicht des C 128. Ein-/Ausschalter, Reset-Knopf, Control-Ports.

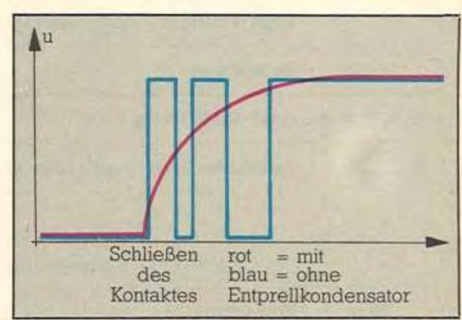


Bild 4. Pegelverlauf beim Prellen eines Tasters

Unser Rundgang nähert sich jetzt immer mehr dem unteren Platinenrand. Dabei sehen wir rechts die Anschlußleiste für die Tastatur. Links davon haben wir einen weiteren wichtigen Baustein, den Complex Interface Adapter CIA 6526, von dessen Sorte übrigens zwei im C 128 eingebaut sind. Die CIA ist ein Input/Output-Baustein. Der Prozessor kann über Bausteine mit seiner übrigen Umwelt in Verbindung treten. Die CIA besitzt zwei, voneinander unabhängige, bitweise auf Ein- oder Ausgang programmierbare Ports, die jeweils 8 Bit »breit« sind. Außerdem sind in einer CIA noch zwei 16-Bit-Timer integriert. Ein serielles Schieberegister fehlt ebensowenig wie eine 24-Stunden-Echtzeituhr.

Mit den Timern lassen sich in erster Linie bequem Interrupts auslösen, es sind jedoch auch andere Anwendungen möglich, zum Beispiel eine Zählung von Impulsen, die über einen Anschluß am User-Port zur CIA gelangen. Das serielle Schieberegister ist im C 128 unbe- Abfrage von Tastatur und Joysticks verwendet und stehen dem Programmierer nicht zur freien Verfügung. Timer A benötigt der C 128 zur Erzeugung des IRQ-Signals. Timer B steht dem Benutzer zur Verfügung, wenn gerade keine Kassetten- oder Diskettenoperation erfolgt. Die IRQ-Ausgangsleitung dieser CIA ist direkt mit der IRQ-Leitung des Prozessors verbunden. Gleich links neben der CIA befinden sich die beiden »Herzhälften« des C 128: die beiden Prozessoren 8502 (links) und Z 80 B (rechts). Die beiden, die ja sonst die gegensätzlichsten Konkurrenten auf dem Heimcomputermarkt sind und sich gegenseitig den Rang abzulaufen versuchen, befinden sich hier vereint nebeneinander auf der Platine des C 128. Vollkommen gleichberechtigt sind die beiden allerdings lange nicht. Außer im CP/M-Modus übernimmt der 8502 alle anfallende Arbeit. Maschinensprache-Programmierer haben allerdings die Möglichkeit, ihre Assemblerprogramme jetzt auch in Z 80-Befehlen fach eingesetzt, vom Sinclair ZX 81 bis zu den großen CP/M-Maschinen. Im Gegensatz zur 65xx-Prozessor-Familie handelt es sich beim Z 80 ebenso wie beim 8080 um eine sogenannte »registerorientierte CPU«. Das bedeutet, daß der Z 80 über eine größere Anzahl interner Register verfügt, die sowohl als Daten- wie auch als Adreßregister benutzt werden können.

Z 80: Sechs 16-Bit-Register

Insgesamt stehen dem Benutzer des Z 80 sechs allgemeine 16-Bit-Register (BC, DE, HL, BC', DE', HL') zur Verfügung, die wahlweise auch als zwölf 8-Bit-Register angesprochen werden können. Daneben gibt es zwei Akkus und ebenfalls zwei Flag-Register, zwei 16-Bit-Indexregister (IX und IY) einen 16-Bit-Stackpointer (SP) und natürlich einen Programmzähler (PC). Zwei weitere Register, das Interrupt-Vektor-Register I und das Refresh-Control-Register R haben spezielle Funktionen. Die Register AF,

Mode Configuration Register	
Bits	**Bedeutung**
7	Stellung des 40/80-Zeichen-Schalters 0 = Schalter geschlossen 1 = Schalter offen
6	Betriebsart 0 = C128 1 = C64
5	Bidirektionaler Port Eingang : /EXROM Ausgang : Farb-RAM-Bank während VIC aktiv
4	Bidirektionaler Port Eingang : /GAME Ausgang : Farb-RAM-Bank während CPU aktiv
3	FSDIR (Fast ser. disk) Kontrollbit Eingang : Fast serial disk enable Ausgang : für FSD-Hardware
2,1	Nicht benutzt
0	CPU-Einstellung 0 = Z80 1 = 8502

Page Pointer, höherwertiges Byte	
Bits	**Bedeutung**
7,6,5,4	ohne Bedeutung
3,2,1,0	gewählte Bank (auch ROM möglich !)

Page Pointer, niederwertiges Byte	
Bits	**Bedeutung**
7 bis 0	Speicherseite innerhalb der 64K-Bank

System Version Register	
Bits	**Bedeutung**
7,6,5,4	verfügbare 64K-Bänke
3,2,1,0	Versionsnummer der MMU

Configuration Register	
Bits	**Bedeutung**
7,6	RAM-Bank-Auswahl 00 = Bank 0 01 = Bank 1 10 = Bank 2 11 = Bank 3
5,4	Speicherbereich $C000-$FFF 00 = SYSTEM ROM (KERNAL, Char. ROM - I/O) 01 = INTERNAL FUNCTION ROM (hi) 10 = EXTERNAL FUNCTION ROM (hi) 11 = RAM
3,2	Speicherbereich $8000-$BFFF 00 = BASIC ROM (hi) 01 = INTERNAL FUNCTION ROM (lo) 10 = EXTERNAL FUNCTION ROM (lo) 11 = RAM
1	Speicherbereich $4000-$7FFF 0 = BASIC ROM (lo) 1 = RAM
0	Speicherbereich $D000-$DFFF 0 = I/O 1 = RAM/ROM (abhängig von Bits 4 und 5)

RAM Configuration Register	
Bits	**Bedeutung**
7,6	RAM-Bank des VIC (Bit 7 für Erweiterung) x0 = RAM-Bank 0 x1 = RAM-Bank 1
5,4	Nicht benutzt (für Erweiterung auf 1 MByte)
3,2	Lage der gemeinsamen Speicherbereiche 00 = keine gemeinsamen Speicherbereiche 01 = Speicheranfang 10 = Speicherende 11 = Speicheranfang und -Ende
1,0	Größe des/der gemeinsamen Speicherbereiche 00 = 1 KByte 01 = 4 KByte 10 = 8 KByte 11 = 16 KByte

Tabelle 1. Die Register der MMU

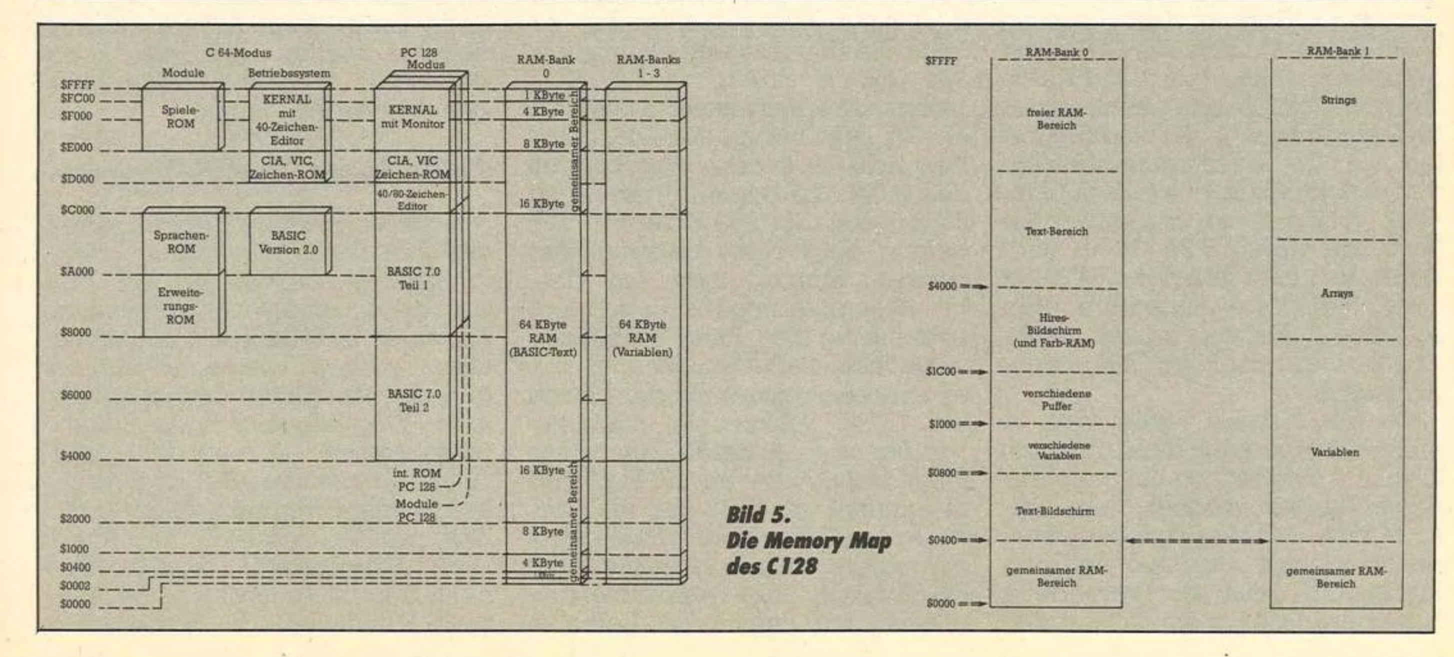


Bild 5. Die Memory Map des C128

Primärregister

A	F
B	C
D	E
H	L

Sekundärregister

A'	F'
B'	C'
D'	E'
H'	L'

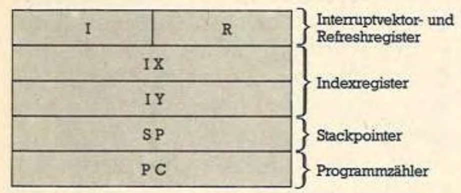
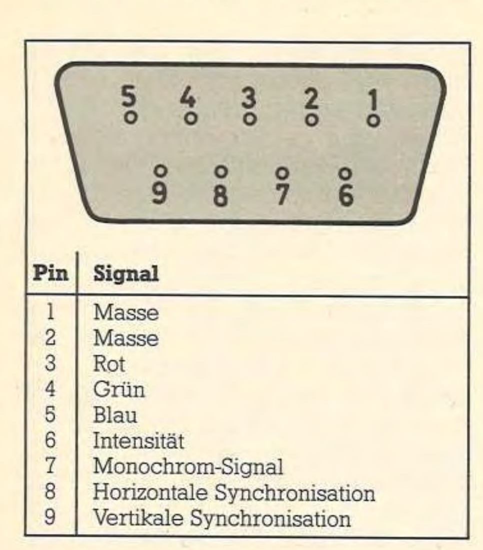


Tabelle 2. Der Z-80-Registersatz

BC, DE und HL bilden die sogenannten Primärregister, die vom 8080 übernommen wurden und die in erster Linie direkt angesprochen werden können. Mit speziellen Befehlen kann zwischen diesen Primärregistern und den Sekundärregistern AF', BC', DE' und HL' umgeschaltet werden. Die Indexregister IX und IY haben ähnliche Aufgaben wie die 6502-Register X und Y, sind jedoch 16 Bit breit. Der Z80-Befehlssatz ist sehr umfangreich und umfaßt unter anderem auch 16-Bit-Arithmetik, Blockverschiebebefehle, automatische Such- und Ein-/Ausgabebefehle, bedingte Unterprogrammaufrufe sowie Einzelbitbefehle. Insgesamt kennt die Z80-CPU über 700 Opcodes. Um diese vielen Maschinenbefehle verschlüsseln zu können (mit einem Befehlsbyte können nur 256 Befehle codiert werden), gibt es einige spezielle Umschalt-Opcodes, die einfach bewirken, daß der Z80 intern auf eine andere Befehlstabelle umschaltet und das nächste Befehlsbyte in einer anderen Form interpretiert. Unser Rundgang führt uns jetzt vorbei an dem Programmable-Logic-Array PLA 8721. Dies ist ein vom Hersteller programmierter Baustein, der eine Fülle von logischen Gliedern wie AND, OR etc. enthält. Mit einem einzigen IC kann man so Logikschaltungen aufbauen, die sonst aus einer großen Anzahl einzelner Logik-ICs bestehen würden. Die Funktion dieses Bausteins ergibt sich im Zusammenhang mit der Memory-Management-Unit. Ebenso wie die MMU ist auch das PLA-IC für die Selektierung der ROMs zuständig.

Unser Weg durch die Hardware, der jetzt ein Stück in Richtung des oberen Randes verläuft, führt uns jetzt an der Stelle der Platine vorbei, die die größte Leiterbahnendichte aufweist. Hier handelt es sich vor allem um Daten- und Adreßleitungen, die zu den RAM-Bausteinen in der linken unteren Ecke der Platine führen. Der C128 hat drei verschiedene Bussysteme (Bild 6). Zum einen existiert der »normale« Adreßbus, der zusammen mit dem Datenbus die verschiedenen ROMs, die beiden Prozessoren, den RGB-Video-Chip, die MMU und die I/O-Bausteine versorgt. Das zweite Bussystem ergibt sich aus dem Multiplexed-Adreß-Bus im Zusammenspiel mit dem Datenbus. Dieses Bussystem sichert die Zusammenarbeit des VICs mit dem Prozessor. Welche Daten über welche Leitungen fließen, bestimmt das MUX-Signal. Es wird von der MMU und der PLA erzeugt. Da die Erklärung der Funktion des Multiplexed-Adreß-Bus hier zu weit führen würde, nur ein Umriß: Er wird vom Prozessor zur Adressierung der beiden RAM-Bänke gebraucht, zur Adressierung der VIC-Register und für Zugriffe des VICs auf das Farb-RAM und das Zeichensatz-ROM. Das dritte Bussystem besteht aus dem Translated Adreßbus und dem Datenbus. Die Funktion des Translated Adreßbus, was soviel bedeutet wie »angepaßter Adreßbus«, ermöglicht das Arbeiten des Z 80 auf dem 8502-Bussystem. Im Z80-Modus wird zu jeder Adresse von Bank 0

Bild 6. Blockschaltbild des C128. Hier sehen Sie, wie die Bauteile der Platine prinzipiell verschalten sind.



Pin	Signal
1	Masse
2	Masse
3	Rot
4	Grün
5	Blau
6	Intensität
7	Monochrom-Signal
8	Horizontale Synchronisation
9	Vertikale Synchronisation

Bild 7. Die Belegung der RGB-Buchse

aus dem Bereich von $0000 bis $0FFF $D000 addiert, damit der Z80 auf das CP/M-BIOS, einem Basisprogramm von CP/M, zugreifen kann. Normalerweise liegt ja bei CP/M das BIOS im Bereich von $D000 bis $DFFF im RAM-Speicher. Da aber die ganze C128-Organisation 8502-spezifisch ist und deshalb das BIOS in Bank 0 von $0000 bis $0FFF untergebracht ist, ist eine Konvertierung der Adressen von $D000 bis $DFFF auf $0000 bis $0FFF nötig. Und zwar immer dann, wenn vom Z80 auf das BIOS zugegriffen wird. Die Adreßkonvertierung findet nur in Bank 0 statt, nicht in Bank 1. Hier befindet sich die normale 8502-System-Zeropage. Der nächste herausragende Baustein ist das, neben der PLA 8721 liegende, 2-KByte-8-Bit CMOS-RAM. Dieser RAM-Baustein ist der Farbspeicher für den VIC-Chip 8564.

Der Farbspeicher ist aus Zeitgründen in einem gesonderten RAM untergebracht. Legt der VIC nämlich beim Auffrischen des Bildschirmes die Adresse an den Adreßbus, an der das momentan darzustellende Zeichen steht, so liegt diese Adresse gleichzeitg am Zeichen-RAM und am Farb-RAM. Weil das Farb-RAM aber einen eigenen Datenbus zum VIC hat (4 Bit, 2^4 (=16) verschiedene Farben), stehen der Bildschirmcode aus dem Video-RAM und die Farbinformation aus dem Farb-RAM gleichzeitig beim VIC bereit. Gleich neben dem Farb-RAM des VIC liegt das 2-KByte-Zeichengenerator-ROM. In diesem ROM befinden sich die Informationen, wie die Zeichen auf dm Bildschirm dargestellt werden sollen, also ob ein Punkt gesetzt wird oder nicht. Der Inhalt dieses ROMs wird beim Umschalten auf den 80-Zeichen-Modus in den RAM-Bereich des VDC 8563 kopiert.

128 KByte in 16 RAMs

Die 16 ICs in der unteren Ecke der Platine bilden zusammen den 128 KByte großen Hauptspeicher des C128. Jeder einzelne dieser Chips hat eine Speicherkapazität von 8 KByte. Da es sich bei den verwendeten RAM-Typen um »dynamische« RAMs handelt, die zuerst die eine Hälfte des Adreßwortes und danach die zweite Hälfte auf den gleichen Anschlüssen verlangen, ist der Multiplexed Adreßbus notwendig. Weiterhin kommen wir jetzt vorbei an den ROMs, in denen Teil 1 und Teil 2 des Basic-Interpreters für den C128-Modus gespeichert sind. Im linken der drei ROMs ist der Basic-Interpreter und das Kernel für den C64-Modus enthalten. Gleich über diesem ROM liegt CIA 2. Für sie gilt natürlich den Registeraufbau betreffend dasselbe wie für CIA 1. Die Basisadresse beträgt hierbei jedoch $DD00. Die Aufgaben von CIA 1 sind aber andere als bei CIA 2. Port A dieser CIA ist zuständig für den Datenverkehr auf dem seriellen Bus, betrifft also hauptsächlich die Datenübertragung Computer – Floppy/Drucker. Eine Ausnahme bilden die Bits 0 und 1. Sie stellen die Adreßbits 14 und 15 für den VIC dar. Zusammen mit den Adreßbits 0 bis 13 des VIC ergibt sich die vollständige Adresse, auf die gerade zugegriffen werden soll. Das bedeutet für den Anwender, daß er den Bereich, auf den der VIC mit seinen 14 Adreßleitungen zugreifen kann (= 16 KByte), innerhalb einer Speicherbank mit 64 KByte an verschiedenen Stellen plazieren kann. Er muß lediglich die Bits 0 und 1 von Register 0 der CIA 2 entsprechend abändern. Die Kombination 00 ergäbe dann, daß der VIC auf die unteren 16 KByte der Bank zugreift (von $0000 bis $3FFF). Bei 01 läge dieser Bereich dann von $4000 bis $7FFF und so weiter. Es ist jedoch zu beachten, daß sich alle Adressierungen des VIC, mit Ausnahme des Farb-RAMs, welches ja ohnehin einen Sonderstatus einnimmt, auf diesen Bereich beziehen, also auch der Zeichengenerator muß dort zu finden sein. Port B der CIA 2 ist für den Benutzer vollständig am User-Port herausgeführt. Ist eine RS232-Schnittstelle aufgesteckt, erfolgt die Steuerung des Datenverkehrs ebenfalls über diesen Port. Die Software dafür ist bereits im Betriebssystem integriert. Die beiden Timer sind auch für die Steuerung einer RS232-Cartridge zuständig, der Benutzer kann über sie jedoch frei verfügen, wenn eine solche Steckkarte nicht angeschlossen ist. Die IRQ-Ausgangsleitung der CIA 2 ist, im Gegensatz zu CIA 1, nicht mit der IRQ-Leitung des Prozessors verbunden, sondern mit seiner NMI-Leitung. Mit der CIA 2 lassen sich also keine IRQs, sondern nur NMIs erzeugen.

Neben der CIA 2 befindet sich noch ein ROM. Hier kann der Benutzer ROMs mit Programmen einstecken, die oft benötigt werden. In der deutschen Version des C128 ist hier der deutsche Zeichensatz untergebracht. Neben diesem ROM befindet sich das 16-KByte Betriebssystem(Kernel)-ROM.

Zwei Video-Chips

Auf unserem Rundgang kommen wir nun an zwei Blechgehäusen vorbei. Die Deckel dieser Gehäuse wurden für das Foto abgenommen. Der obere Blechkasten schirmt den HF-Modulator ab, der aus dem Video-Signal des VIC-Chips ein für Fernseher »verständliches« Signal macht, das über die Antennenbuchse einem Ferseher zugeführt werden kann. Im unteren Gehäuse

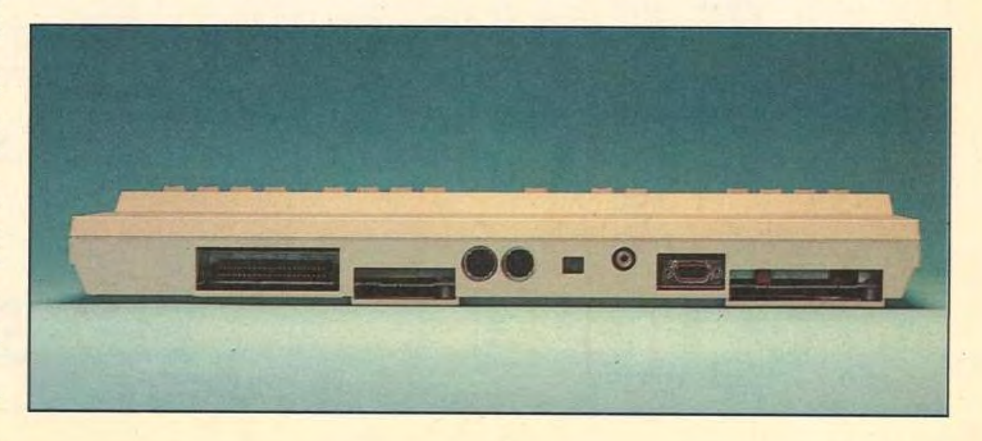
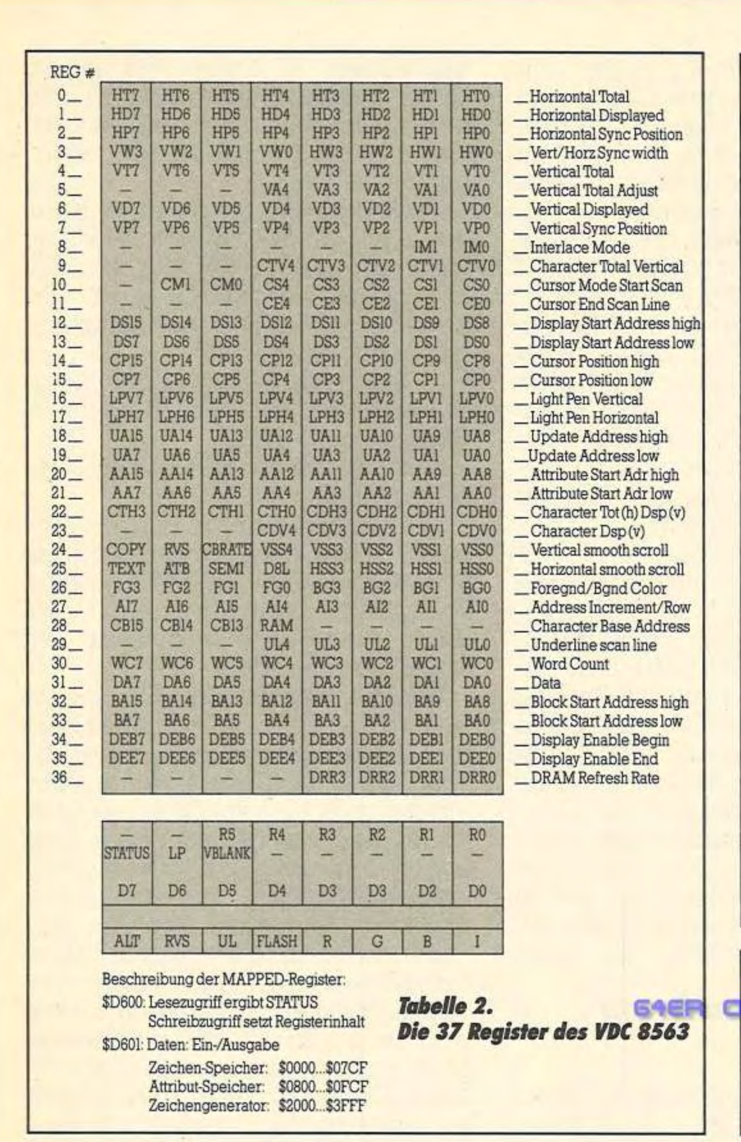


Bild 8. Die Rückseite des C128



REG #									
0	HT7	HT6	HT5	HT4	HT3	HT2	HT1	HT0	Horizontal Total
1	HD7	HD6	HD5	HD4	HD3	HD2	HD1	HD0	Horizontal Displayed
2	HP7	HP6	HP5	HP4	HP3	HP2	HP1	HP0	Horizontal Sync Position
3	VW3	VW2	VW1	VW0	HW3	HW2	HW1	HW0	Vert/Horz Sync width
4	VT7	VT6	VT5	VT4	VT3	VT2	VT1	VT0	Vertical Total
5	—	—	—	VA4	VA3	VA2	VA1	VA0	Vertical Total Adjust
6	VD7	VD6	VD5	VD4	VD3	VD2	VD1	VD0	Vertical Displayed
7	VP7	VP6	VP5	VP4	VP3	VP2	VP1	VP0	Vertical Sync Position
8	—	—	—	—	—	—	IM1	IM0	Interlace Mode
9	—	—	—	CTV4	CTV3	CTV2	CTV1	CTV0	Character Total Vertical
10	—	CM1	CM0	CS4	CS3	CS2	CS1	CS0	Cursor Mode Start Scan
11	—	—	—	CE4	CE3	CE2	CE1	CE0	Cursor End Scan Line
12	DS15	DS14	DS13	DS12	DS11	DS10	DS9	DS8	Display Start Address high
13	DS7	DS6	DS5	DS4	DS3	DS2	DS1	DS0	Display Start Address low
14	CP15	CP14	CP13	CP12	CP11	CP10	CP9	CP8	Cursor Position high
15	CP7	CP6	CP5	CP4	CP3	CP2	CP1	CP0	Cursor Position low
16	LPV7	LPV6	LPV5	LPV4	LPV3	LPV2	LPV1	LPV0	Light Pen Vertical
17	LPH7	LPH6	LPH5	LPH4	LPH3	LPH2	LPH1	LPH0	Light Pen Horizontal
18	UA15	UA14	UA13	UA12	UA11	UA10	UA9	UA8	Update Address high
19	UA7	UA6	UA5	UA4	UA3	UA2	UA1	UA0	Update Address low
20	AA15	AA14	AA13	AA12	AA11	AA10	AA9	AA8	Attribute Start Adr high
21	AA7	AA6	AA5	AA4	AA3	AA2	AA1	AA0	Attribute Start Adr low
22	CTH3	CTH2	CTH1	CTH0	CDH3	CDH2	CDH1	CDH0	Character Tot (h) Dsp (v)
23	—	—	—	CDV4	CDV3	CDV2	CDV1	CDV0	Character Dsp (v)
24	COPY	RVS	CBRATE	VSS4	VSS3	VSS2	VSS1	VSS0	Vertical smooth scroll
25	TEXT	ATB	SEMI	D8L	HSS3	HSS2	HSS1	HSS0	Horizontal smooth scroll
26	FG3	FG2	FG1	FG0	BG3	BG2	BG1	BG0	Foregnd/Bgnd Color
27	AI7	AI6	AI5	AI4	AI3	AI2	AI1	AI0	Address Increment/Row
28	CB15	CB14	CB13	RAM	—	—	—	—	Character Base Address
29	—	—	—	UL4	UL3	UL2	UL1	UL0	Underline scan line
30	WC7	WC6	WC5	WC4	WC3	WC2	WC1	WC0	Word Count
31	DA7	DA6	DA5	DA4	DA3	DA2	DA1	DA0	Data
32	BA15	BA14	BA13	BA12	BA11	BA10	BA9	BA8	Block Start Address high
33	BA7	BA6	BA5	BA4	BA3	BA2	BA1	BA0	Block Start Address low
34	DEB7	DEB6	DEB5	DEB4	DEB3	DEB2	DEB1	DEB0	Display Enable Begin
35	DEE7	DEE6	DEE5	DEE4	DEE3	DEE2	DEE1	DEE0	Display Enable End
36	—	—	—	—	DRR3	DRR2	DRR1	DRR0	DRAM Refresh Rate

— STATUS	— LP	R5 VBLANK	R4 —	R3 —	R2 —	R1 —	R0 —
D7	D6	D5	D4	D3	D3	D2	D0
ALT	RVS	UL	FLASH	R	G	B	I

Beschreibung der MAPPED-Register:

$D600: Lesezugriff ergibt STATUS
Schreibzugriff setzt Registerinhalt

$D601: Daten: Ein-/Ausgabe

Zeichen-Speicher: $0000...$07CF
Attribut-Speicher: $0800...$0FCF
Zeichengenerator: $2000...$3FFF

Tabelle 2. Die 37 Register des VDC 8563

REG#									
0	S0x7	S0x6	S0x5	S0x4	S0x3	S0x2	S0x1	S0x0	Sprite 0 X Koordinate
1									Sprite 0 Y Koordinate
2									Sprite 1 X Koordinate
3									Sprite 1 Y Koordinate
4									Sprite 2 X Koordinate
5									Sprite 2 Y Koordinate
6									Sprite 3 X Koordinate
7									Sprite 3 Y Koordinate
8									Sprite 4 X Koordinate
9									Sprite 4 Y Koordinate
10									Sprite 5 X Koordinate
11									Sprite 5 Y Koordinate
12									Sprite 6 X Koordinate
13									Sprite 6 Y Koordinate
14									Sprite 7 X Koordinate
15									Sprite 7 Y Koordinate
16	S7x8	S6x8	S5x8	S4x8	S3x8	S2x8	S1x8	S0x8	Sprite X plus 256
17	RCB	ECM	BMM	Blnk	ROWS	Y2	Y1	Y0	Modus, Y-scroll
18	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	Raster-Register
19	LPx8	LPx7	LPx6	LPx5	LPx4	LPx3	LPx2	LPx1	Lightpen X-Koordinate
20	LPy7	LPy6	LPy5	LPy4	LPy3	LPy2	LPy1	LPy0	Lightpen Y-Koordinate
21	SD7	SD6	SD5	SD4	SD3	SD2	SD1	SD0	Sprite X: AN/AUS
22	--	--	Rset	MCM	COLS	X2	X1	X0	Modus, X-Scroll
23	S7Ey	S6Ey	S5Ey	S4Ey	S3Ey	S2Ey	S1Ey	S0Ey	Sprite in Y-Richtung vergrößern
24	VM13	VM12	VM11	VM10	CB13	CB12	CB11	--	Basisadresse BSS und ZG *
25	IRQ	--	--	--	LP	S/S	S/B	RIRQ	Interrupt-Flags
26	--	--	--	--	_LP	_S/S	_S/B	_IRQ	Interrupt-Maske
27	BSP7	BSP6	BSP5	BSP4	BSP3	BSP2	BSP1	BSP0	Sprite-Hintergr. Priorität
28	MCS7	MCS6	MCS5	MCS4	MCS3	MCS2	MCS1	MCS0	Sprite X=Multicolor
29	S7Ex	S6Ex	S5Ex	S4Ex	S3Ex	S2Ex	S1Ex	S0Ex	Sprite in X-Richtung vergrößern
30	SS7	SS6	SS5	SS4	SS3	SS2	SS1	SS0	Kollision Sprite/Sprite
31	SB7	SB6	SB5	SB4	SB3	SB2	SB1	SB0	Kollision Sprite/Hintergr. (Zeichen)
32	--	--	--	--					Randfarbe
33	--	--	--	--					Hintergrundfarbe 1
34	--	--	--	--					Hintergrundfarbe 2
35	--	--	--	--					Hintergrundfarbe 3
36	--	--	--	--					Hintergrundfarbe 4
37	--	--	--	--					Sprite Mehrfarbregister 1
38	--	--	--	--					Sprite Mehrfarbregister 2
39	--	--	--	--					Sprite 0 Farbe
40	--	--	--	--					Sprite 1 Farbe
41	--	--	--	--					Sprite 2 Farbe
42	--	--	--	--					Sprite 3 Farbe
43	--	--	--	--					Sprite 4 Farbe
44	--	--	--	--					Sprite 5 Farbe
45	--	--	--	--					Sprite 6 Farbe
46	--	--	--	--					Sprite 7 Farbe
47	--	--	--	--	--	K2	K1	K0	Tastatur-Kennung K2..K0
48	--	--	--	--	--	--	TEST	2MHz	Takt-Geschwindigkeit

* BGS = Bildschirmspeicher
ZG = Zeichengenerator

Tabelle 3. Die Register des VIC 8564

Pin	Signal	Pin	Signal
1	LUMINANCE	6	CROMINANZ
2	GND	7	O. BELEGUNG
3	AUDIO OUT	8	O. BELEGUNG
4	VIDEO OUT		
5	AUDIO IN		

Bild 9. Die Belegung der Video/Audio-Buchse

sind der RGB- und der PAL-Video-Teil untergebracht. Ganz links ist der Video-Controller VDC 8563 (RGB-Buchse zeigt Bild 7, 8) zu erkennen. Der VIC 8564 (Videobuchse zeigt Bild 9) sitzt der rechten Hälfte des Gehäuses. Diese beiden Video-Systeme sind genauso gegensätzliche Konkurrenten wie der 8502 und der Z80. Der VDC 8563, der im Gegensatz zum VIC 8564 in der Lage ist, 80 Zeichen pro Zeile darzustellen, wurde im C128 aus zwei Gründen eingesetzt. Erstens, weil ein CP/M-fähiger Computer mit nur 40 Zeichen pro Zeile vergleichbar wäre mit einem Formel 1-Rennwagen, der durch einen Rasenmähermotor angetrieben wird. Mit 40 Zeichen pro Zeile ist keine professionelle Textverarbeitung oder Tabellenkalkulation zu realisieren.

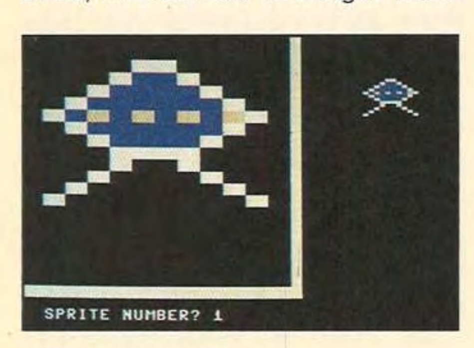


Bild 10. Der C128 hat sogar einen eingebauten Sprite-Editor

Der zweite und gewichtigere Grund für die Verwendung des VDC 8563 ist der, daß der VIC 8564 bei einer Taktfrequenz von über 1 MHz »nicht mehr mitkommt«. Das würde für den Rennwagen bedeuten, nun gänzlich ohne Antrieb dazustehen. Manche werden sich sicherlich fragen, wozu denn eigentlich der VIC dann noch da ist. Die Antwort fällt nicht schwer: Ohne den VIC wäre der C128 nicht mehr voll kompatibel zum C64. Daß der VIC bei höheren Taktfrequenzen nicht mehr mitkommt liegt daran, daß er die Taktlücken des Prozessors ausnutzt, um seine Bildwiederholinformation aus dem Speicher zu holen. Mit steigender Frequenz werden diese Taktlücken natürlich kürzer, und damit zu kurz für den VIC. Der VDC besitzt aber ein eigenes, 16 KByte großes RAM und ist damit nicht mehr auf die Taktlücken des Prozessors angewiesen. In diesem Speicherbereich sind für den 8563 alle Daten gespeichert. Dieses RAM ist jedoch nicht im Speicherbelegungsplan des C128 zu finden, sondern direkt an den VDC angeschlossen. Ansprechbar ist dieses RAM indirekt über zwei Register des VDC.

Der VDC 8563 besitzt 37 Steuerregister (Tabelle 2), deren Beschreibung den Rahmen dieses Artikels sprengen würde. Interessant ist jedoch, daß die Register des VDC nicht einfach direkt anzusprechen sind wie beispielsweise bei der CIA oder dem VIC. Es reicht also nicht, einfach die Basisadresse (beim VDC $D600) und die Registernummer zu addieren. Die Adressierung der Register erfolgt relativ. Man schreibt in die Adresse $D600 die Nummer des angesprochenen Registers und erhält im Gegenzug vom VDC in Adresse $D601 den Inhalt des angesproche-

nen Registers. Soll der Inhalt eines Registers geändert werden, schreibt man in Adresse $D601 den betreffenden Wert ein, nachdem das Register mit $D600 spezifiziert wurde. Will man zum Beispiel eine Adresse des VDC-eigenen RAMs ändern, verfährt man folgendermaßen:

1. POKE 54784,18 – als Register wird 18 ausgewählt. Register 18 und 19 bestimmen im High/Low-Byte-Format die angesprochene RAM-Adresse.

2. POKE 54785, (Adresse High-Byte) – das High-Byte der Adresse wird in Register 18 geschrieben

3. POKE 54784,19 – Auswahl: Register 19

4. POKE 54785, (Adresse Low-Byte) – das Low-Byte der Adresse kommt nach Register 19

5. POKE 54584,31 – jetzt wird Register 31 angesprochen. Ins Register 31 wird der Wert geschrieben, der im Speicher abgelegt werden soll.

6. POKE 54585, (Wert) – man sieht, die Handhabung ist etwas umständlich und gewöhnungsbedürftig. Im VDC 8563 sind nicht, wie im VIC, alle Register für den Programmierer nutzbringend. Viele werden zum Beispiel für Synchronisationszwecke benutzt und sollten oder dürfen vom Programmierer nicht verändert werden. Es steckt aber trotzdem eine schier unüberschaubare Vielzahl von Variationsmöglichkeiten in diesem Chip, denen man nur durch Ausprobieren auf die Spur kommen kann. Der VDC 8563 gestattet sogar eine hochauflösende Grafik mit 640 (!) mal 200 Punkten. Die Grafikbefehle des Basic 7.0 sind hier jedoch nicht wirksam. Entsprechende Programme sind in der einschlägigen Literatur zu finden. Allerdings verlangt der VDC 8563 nach einem RGB-Monitor, will man sich nicht mit Schwarz/Weiß begnügen. Dies ist auch der Grund dafür, daß man für den C 128 eigentlich zwei Monitore braucht: Einen RGB-Monitor für den 8563 und einen Composite-Monitor oder einen Fernseher für den 8564. Es gibt aber inzwischen genügend Monitore, die mit diesen beiden Normen arbeiten können.

1 MHz oder 2 MHz – Wie hätten Sie es gern?

Einfacher als beim VDC 8563 gestaltet sich die Programmierung des VIC 8564, der ja schon hinlänglich aus dem C 64 bekannt ist. Es sind jedoch noch zwei Steuerregister hinzugekommen. Register 48 ist zuständig für die dem C 64 gegenüber erweiterte Tastatur. Register 49 bestimmt, ob sich der C 128 im 1-MHz- oder 2-MHz-Modus befindet. Um etwa die Rahmenfarbe auf Schwarz zu schalten, gibt man einfach die Befehlsfolge »POKE 53280,0« ein. »0« ist der Farbcode für Schwarz. Die Adresse 53280 ergibt sich, wenn man zur Basisadresse des VIC ($D000 oder 53248) die Registernummer für die Rahmenfarbe (Register 32) addiert. Dies funktioniert natürlich auch mit allen anderen Farben, die der VIC darstellen kann (Tabelle 3).

Die Auflösung des VIC 8564 beträgt in X-Richtung gerade die Hälfte der des VDC 8563. Der VIC kann horizontal 40 Zeichen darstellen. Die Zeilenzahl beträgt 25. Bei einer Punktmatrix von 8x8 Punkten bedeutet das für den HiRes-Bildschirm eine Auflösung von 320x200 Punkten. Allerdings sind in dieser Auflösung nur zweifarbige Grafiken möglich, nämlich in der Hintergrundfarbe und der Punktfarbe. Mehrfarbige Grafiken können im Multicolor-Modus erzeugt werden, wobei dann allerdings die Auflösung von 320 Punkten in horizontaler Richtung auf 160 sinkt. Möglich sind jetzt allerdings fünf Farben, die Hintergrundfarbe sowie vier verschiedene Punktfarben. Auf diesen Grafik-Bildschirm beziehen sich auch die Grafikbefehle des Basic 7.0, wie etwa »Circle« oder »Graphic«.

Sprites – Kobolde am Bildschirm

Ein weiterer großer Vorteil des VIC gegenüber dem VDC sind die durch ihn verwalteten Sprites (engl. »Kobolde«). Sprites sind kleine Grafikgebilde mit einer Auflösung von 24x21 Punkten, die der Benutzer selbst programmiert. Mit diesen Sprites kann man allerhand anfangen. Man kann sie zum Beispiel frei über den gesamten Bildschirm bewegen, oder sie in X- oder Y-Richtung vergrößern. Sprites sind sogar im Multicolor-Modus darstellbar, allerdings auch nur mit einer geringeren Auflösung von 16x21 Punkten. Der C 128 hat einen Editor zum Erstellen von Sprites (Bild 10).

Sehr bequem macht es der VIC den Spiele-Programmierern. Will man zum Beispiel feststellen, ob das Raumschiff des Spielers mit irgendeinem Asteroiden kollidiert ist, kann dies über zwei Register geschehen, in denen für jeden Sprite getrennt aufgeführt ist, ob bereits eine Sprite-Sprite- oder eine Sprite-Hintergrund-Kollision stattgefunden hat.

Über ein weiteres Register kann der Programmierer die Priorität festlegen, in der ein Sprite auf dem Bildschirm dargestellt wird. Es bestehen zwei Variationsmöglichkeiten für die Priorität:

1. Das Sprite verschwindet hinter den Zeichen auf dem Bildschirm (Hintergrund) oder

2. das Sprite überdeckt den Bildschirminhalt.

Die Priorität der Sprites untereinander ist so geregelt, daß der Sprite mit der kleinen Nummer die höhere Priorität hat. Sprite 4 verdeckt also Sprite 5 und wird seinerseits von Sprite 3 verdeckt.

Weitere Register im VIC legen fest, ob der Sprite in X-, Y- oder beiden Richtungen vergrößert dargestellt wird, und zwar wieder für jeden Sprite einzeln. Über ein spezielles Register ist der VIC in der Lage, Interrupts auszulösen. Er kann den Prozessor mitten in seiner Arbeit unterbrechen und ihm mitteilen, daß dies oder jenes Ereignis eingetreten ist, wozu auch die Sprite-Kollisionen gehören. Der VIC kann aber auch einen sogenannten Rasterzeilen-Interrupt auslösen. Dieser Interrupt wird jedesmal dann erzeugt, wenn der Zeilenstrahl des Monitors eine bestimmte Zeile passiert hat. Man kann diesen Effekt dazu benutzen, hochauflösende Grafik mit einem Textbereich zu versehen, indem man zum Zeitpunkt des Rasterinterrupts vom HiRes-Modus auf den Textmodus schaltet. Der Grafikmodus muß natürlich auch wieder ab einer bestimmten Zeile eingeschaltet werden.

Interessant ist ebenfalls, daß das gesamte Timing des C 128 im 1 MHz-Modus vom VIC gesteuert wird. Nur so ist jederzeit gewährleistet, daß der VIC nicht die Taktlücken des Prozessors verpaßt, in denen er seine Bildwiederholinformationen aus dem Speicher beziehen kann.

Auch das Refresh der dynamischen RAMs wird komplett vom VIC übernommen, auch im 2 MHz-Modus. An seinem Ausgang stellt der VIC ein normiertes PAL- und Composite-Signal bereit. Dieses Signal wird von jedem Monitor mit Video-Composite-Eingang verstanden. Es kann aber auch jeder Fernseher über den eingebauten HF-Modulator angeschlossen werden. Allerdings leidet die Bildqualität



dann schon erheblich. Das Signal muß ja schließlich erst im HF-Modulator moduliert und nachher im Empfangsteil des Fernsehers wieder demoduliert werden.

Bewährt: SID 6581

Ein weiterer alter Bekannter aus dem C 64 verrichtet seine Dienste gleich nebenan. Es handelt sich um den SID 6581 (SID = Sound-Interface-Device). Der SID bietet eine solche Fülle von Kombinationsmöglichkeiten, daß nur durch Ausprobieren ein Überblick über die Funktionen des SID gewonnen werden kann. Auch die Programmierung des SID wird durch das Basic 7.0 des C 128 unterstützt. Zu den besten Klangergebnissen kommt man aber immer noch durch Maschinensprache.

Die Funktionen des SID werden über 27 Register, die direkt adressierbar sind, gesteuert. Die Basisadresse ist $D400. Im SID sind drei verschiedene Oszillatoren untergebracht, die vollkommen unabhängig voneinander funktionieren. Für jeden Oszillator kann als Wellenform zwischen Dreieck, Sägezahn, Rechteck mit variablem Pulsverhältnis und weißem Rauschen gewählt werden. Das überstreichbare Frequenzspektrum reicht von 0 bis 8,2 kHz. Für jeden Oszillator gibt es zwei Register zur Frequenzwahl. Der in diese Register einzuschreibende Wert entspricht allerdings nicht der Frequenz, sondern ist ihr nur proportional. Für die folgenden Parameter existieren auch Register. Die Lautstärke kann in 16 Abstufungen von 0 bis 15 geregelt werden, allerdings nur für das, den SID verlassende, Mischsignal.

Die sogenannte ADSR-Funktion (Bild 11) steht für Attack, Decay, Sustain und Release, was soviel bedeutet wie Anschwellen, Abfallen, Halten und Ausklingen. Diese Funktion bestimmt die Lautstärke und Dauer der Vorgänge. Jeder einzelne Parameter kann einen Wert von 0 bis 15 annehmen und ergibt für die Attack-Spanne Zeiten von 0,002 bis 8 Sekunden. Die Decay/Release-Dauer beträgt zwischen 0,006 und 24 Sekunden.

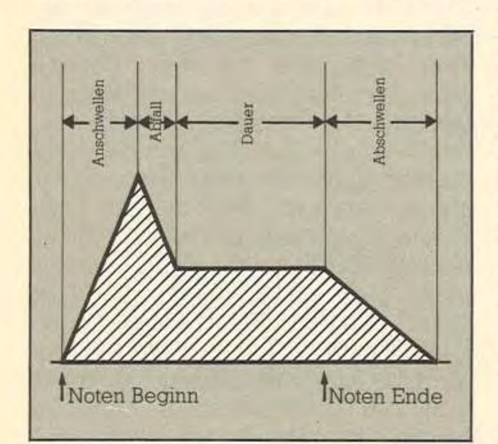


Bild 11. Die ADSR-Kurve des SID-Chips. Die Dauer der Attack-, Decay-, Sustain und Release-Phase können in 15 Abstufungen verändert werden.

Den Oszillatoren ist ein Filter mit wählbarer Hoch-, Tief- und Bandpaß-Eigenschaft nachgeschaltet. Die Grenzfrequenzen sind wählbar. An den Eingang dieses Filters ist auch eine Leitung mit der Bezeichnung »Ext. In« angeschlossen, welche an der Audio/Video-Buchse herausgeführt ist. An diese Leitung kann ein externes Tonsignal angelegt werden, das durch den SID verfremdet werden kann. Das Geräusch kommt dann aus dem Lautsprecher des Fernsehers oder Monitors, sofern dieser einen Audio-Teil besitzt.

Wichtig für den Programmierer ist zu wissen, daß aus all diesen Registern mit PEEK nicht gelesen werden kann. Sollen Toneffekte programmiert werden, die auf den vorherigen Parametern aufbauen, muß der Programmierer diese zuvor zwischenspeichern. Eine besondere Stellung unter den drei Stimmen nimmt die Stimme 3 ein. Über zwei spezielle Register kann hier jederzeit die Hüllkurve und der Oszillatorzustand festgestellt werden. Der Wert aus dem Hüllkurven-Register ist proportional der momentanen Amplitude der ADSR-Funktion. Der Wert aus dem Oszillator-Register ist proportional der Momentan-Amplitude der gewählten Wellenform. Mit diesen Möglichkeiten kann man besonders bequem Zufallszahlen von Assemblerprogrammen aus generieren. Die Kurvenform sollte Rauschen sein. In erster Linie sind mit diesen beiden Registern sehr eindrucksvolle Toneffekte zu erzeugen, indem die Parameter der anderen Oszillatoren (zum Beispiel Frequenz, ADSR-Funktion) oder die Filterfrequenz in Abhängigkeit von diesen Registerinhalten gesetzt werden.

SID-Chip – Sound ist nicht alles

Neben den Sound-Möglichkeiten sind im SID 6581 noch zwei Analog/Digital-Konverter (kurz ADC) integriert. Ein ADC ist eine Schaltung, mit der ein analoger Wert, zum Beispiel eine Spannung, in einen digitalen Zahlenwert überführt wird. Das Problem der Analog/Digital-Wandlung besteht darin, daß man einen analogen Wert, der eine eigentlich unendlich kleine Abstufung besitzt, in einen digitalen Wert mit fest vorgegebenen Intervallen umwandeln muß. Der dabei entstehende Wandlungsfehler liegt bei einem kleinsten digitalen Schritt. Die ADCs im SID sind aber so ungenau und unlinear, daß sie von vornherein für genaue, quantitative Messungen ausscheiden. Bei der Anwendung für die Paddles fällt dies jedoch nicht sonderlich ins Gewicht. Bei der Messung wird über einen veränderlichen Widerstand (im Paddle ist dies ein Potentiometer) ein Kondensator aufgeladen. Gleichzeitig wird mit Meßbeginn eine Uhr gestartet. Erreicht die Spannung des Kondensators eine bestimmte Referenzspannung, wird der Uhr-Stand in ein Register übernommen und ist ein direktes Maß für den Widerstand. Jetzt wird sofort der Kondensator entladen und die Uhr erneut gestartet. Das Meßverfahren beginnt von vorne. An den C 128 sind vier Paddles anschließbar (zwei Paare). Der SID besitzt aber nur zwei Wandler. Die Leitungen mußten also »gemultiplexed« werden. Die Auswahl des aktuellen Paddle-Paares geschieht über Bit 6 und 7 von Port A der CIA 1.

Schnittstellen zur Außenwelt

Zum Abschluß unseres kleinen Rundganges durch die Hardware des Commodore 128 bleibt noch etwas zu den Verbindungen des C 128 zu seiner Umwelt zu sagen. Wie der C 64 hat der C 128 einen Expansion-Port, der von beiden Betriebsmodi angesteuert wird. Im C 64-Modus entspricht die Pinbelegung genau der des C 64. Im Gegensatz zum C 64 erlaubt der Expansion-Port des C 128 einen direkten Speicherzugriff (DMA, Direct Memory Access). Direkter Speicherzugriff bedeutet, daß ohne Umwege über den Prozessor in den Speicher des C 128 geschrieben oder der Speicher ausgelesen werden kann. Das wichtigste, um DMA realisieren zu können, ist, daß der Prozessor während des Zugriffs abgeschaltet bleibt. Beim C 128 macht das der VIC 8564. Er steuert den Daten- und Adreßbus so, daß Prozessor und DMA sich nicht ins Gehege kommen, was beim C 64 nicht immer sichergestellt ist. Bei

Fortsetzung auf Seite 21

Der C 128 D im ersten Test



Wie ein PC sieht er aus, der C 128 D. Elegant, platzsparend und professionell. Er macht Schluß mit Kabelsalat und anderen Ärgernissen. Doch wie läßt sich mit ihm arbeiten?

Bild 1. Die Abschirmung des C 128 D ist fast zu gut gelungen, denn ein Service-Techniker muß ein paar Dutzend Schrauben lösen, bis er die Platinen endlich sieht.

Der C 128 im professionellen PC-Look, so präsentiert sich Commodores jüngstes Kind, der C 128 D. Das D steht für »Disk«, denn der C 128 D hat gleich ein Diskettenlaufwerk eingebaut. Einen Spitznamen hat er auch schon, denn manche Leute interpretieren das »D« als Abkürzung für Diesel ...

Keine Angst, der C 128 D läuft mit der ganz normalen Netzspannung von 220 Volt. Und hat man den C 128 D an eine Steckdose angeschlossen und eingeschaltet, meldet er sich genauso wie ein echter C 128. Ist er's aber auch? Denn viele werden sich noch mit Schrecken an den »vollkompatiblen« SX 64 erinnern, der erhebliche Schwierigkeiten mit so manchem Programm hatte, das der C 64 problemlos verarbeitete. Aber keine Angst, beim C 128 D hat sich eigentlich nur das Äußere geändert. Schraubt man ihn auf, findet man die nur minimal veränderten Platinen des C 128 und der 1571 wieder. Diese sind allerdings fantastisch verpackt, wie in Bild 1 zu sehen. Commodore hat sich noch nie soviel Mühe mit der Abschirmung der einzelnen Baugruppen gegeben wie beim C 128 D. So kommt man, will man den Computer nicht komplett auseinandernehmen, gar nicht erst an die Platinen heran. Die zahlreichen Abschirmbleche zeichnen sich nämlich durch besonders gute Verschraubung mit dem Gehäuse aus. Ein kurzer Blick in die Betriebssysteme

von Computer und Laufwerk erweckte den Eindruck, daß hier überhaupt keine Änderungen vorgenommen wurden. Sollte sich dies bewahrheiten, würde das heißen, daß der C 128 D nicht nur kompatibel, sondern identisch mit dem normalen C 128 ist. Dies beweisen auch die identischen Anschlüsse, die wir in Bild 2 und 3 dargestellt haben. Für die abgesetzte Tastatur mußte nur eine weitere Buchse hinzugefügt werden.

Aufgrund des integrierten Floppy-Laufwerks und der abgesetzten Tastatur läßt sich mit dem C 128 D hervorragend arbeiten. Die Tastatur entspricht vollkommen der des C 128, sie wurde bloß in ein eigenes Gehäuse verfrachtet. Zu Transport- und Aufräumzwecken kann die Tastatur unter dem C 128 D festgeklinkt werden (Bild 4). Der gesamte Computer läßt sich dann an einem ausklappbaren Handgriff durch die Lande tragen. Allerdings: Der C 128 D hat ja keinen eingebauten Monitor, also kann man ein funktionsfähiges System gar nicht mal so einfach transportieren. In einer Hand den Computer, in der anderen den Monitor – diesen Zustand machen nur Bodybuilder längere Zeit mit.

Aber das Hauptargument für den C 128 D ist ja nicht sein Handgriff. Denn wer das Gerät auf dem Schreibtisch stehen hat, der erspart sich gegenüber dem C 128 das externe Netzteil, das Netzkabel für die Floppy und das serielle Buskabel. Außerdem hat er eine frei bewegliche Tastatur an einem Spiralkabel und kann den Monitor platzsparend direkt auf den Computer stellen (Die Abschirmung macht's möglich). Ein richtig professioneller Arbeitsplatz, der sich nur durch sein Innenleben von einem Commodore PC 10 oder einem IBM-PC unterscheidet. Denn der C 128 D ist wohl der einzige Computer, der sich im Büro gleichermaßen für Textverarbeitung und Datenverwaltung (»Wordstar« und »dBase II« im CP/M-Modus, Protext und Superscript für den C 128-Modus), Programmierung (Basic 7.0 im C 128-Modus) und Unterhaltung (»Impossible Mission«, »Summer Games« und über 1000 andere Spiele im C 64-Modus) eignet.

Der C 128 D soll in den Handel kommen, wenn dieses Sonderheft gerade an den Kiosken liegt. Was ihn besonders interessant macht: Er soll etwa hundert Mark billiger sein als das Team C 128/1571. Sollte sich diese vorläufige und unverbindliche Preisauskunft bewahrheiten, bleibt die Frage offen, wer sich dann noch den einfachen C 128 kaufen wird. Denn ein optisch schöneres aber technisch identisches Gerät für weniger Geld — wer würde da nicht zuschlagen? (bs)

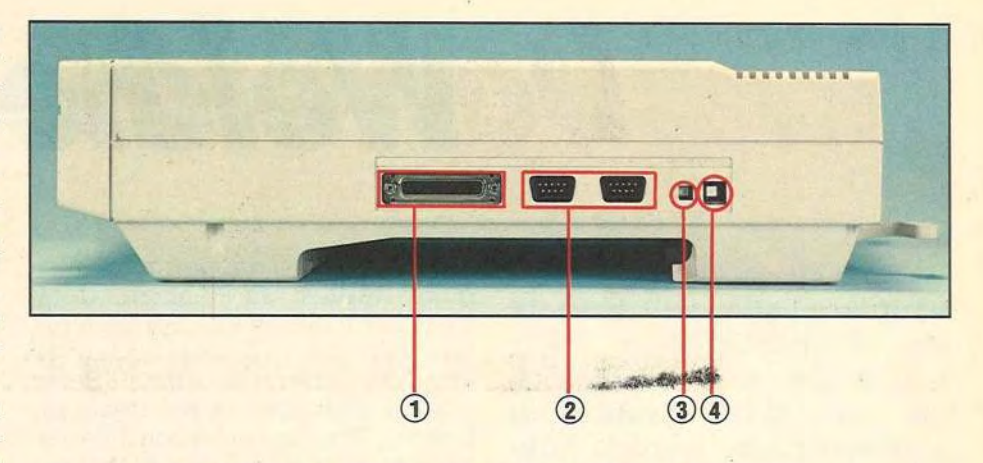


Bild 2. Die Anschlüsse des C 128 an der linken Seite: 1-Tastatur; 2-Joystickports; 3-Resettaste; 4-Netzschalter

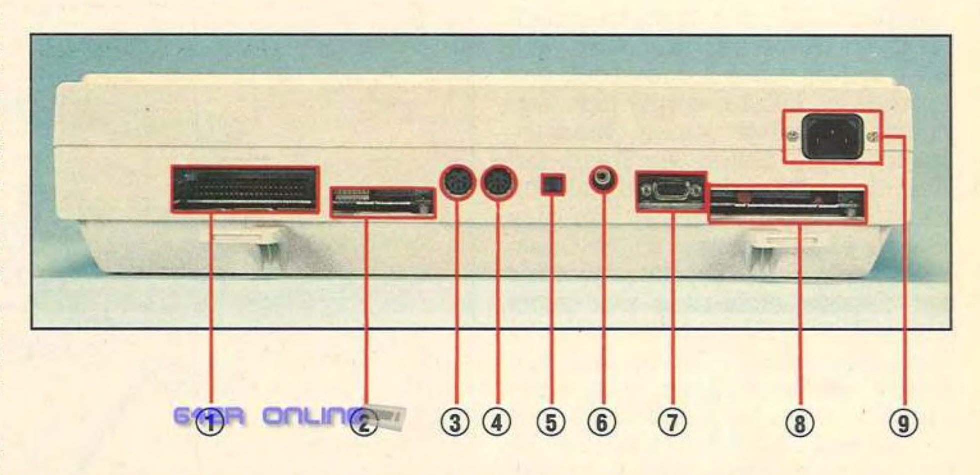


Bild 3. Die Anschlüsse des C 128 an der Rückseite: 1-Expansion-Port; 2-Datasettenanschluß; 3-Serieller Bus; 4-Videoausgang; 5-Kanalwahl (nicht bei deutschen Versionen); 6-Antennenausgang; 7-RGB-Anschluß; 8-User-Port; 9-Netzkabel



Bild 4. Die Tastatur läßt sich bequem an der Unterseite des C 128 D verstauen.

Der C 128 D auf einen Blick:

Computer: C 128
- 128 KByte RAM
- Prozessoren: 8510, Z 80
- Basic 7.0
- CP/M-fähig
- C 64-kompatibel
- abgesetzte Tastatur mit Funktionstasten und Zehnerblock
- Monitore: Composite – (40 Zeichen) RGB – (80 Zeichen)
- Antennenausgang
- hochauflösende Farbgrafik (160 x 200) mit Basic-Befehlen
- dreistimmiger Sound-Chip

Laufwerk: 1571
- Kapazität: 170–400 KByte
- bearbeitet beide Diskettenseiten
- verarbeitet GCR- und MFM-Formate (liest und schreibt 1541-, IBM-, Osborne-, Kaypro- etc. Disketten)

Welche Floppy

Wer mit einem Computer vernünftig arbeiten will, braucht ein Floppylaufwerk. Wir haben die drei Laufwerke, die am C128 problemlos angeschlossen werden können, getestet.

Der C128 soll ein professioneller Computer sein, der auch im Büro oder Kleinbetrieb einsetzbar ist. Dazu benötigt man verständlicherweise einen Massenspeicher, um Daten wie Texte und Kalkulationen oder Kundenlisten dauerhaft zu speichern. Da das Ganze einigermaßen schnell sein muß, kommt ein Kassettenrecorder zur Datenspeicherung gar nicht erst in Frage. Also muß ein Diskettenlaufwerk zum C128 her. Nun fällt dem C128-Besitzer die Wahl recht einfach, denn augenblicklich gibt es nur drei Diskettenlaufwerke, die sich völlig problemlos an den C128 anschließen lassen: die 1541, die 1570 und die 1571. Fairerweise muß gesagt werden, daß es für die 1541 inzwischen schon Ersatztypen von verschiedenen Herstellern gibt. Allerdings sind diese Ersatz-Laufwerke nicht völlig problemlos, da aus urheberrechtlichen Gründen das interne Betriebssystem der Fremd-Laufwerke ganz anders aufgebaut ist. So kann es zu erheblichen Schwierigkeiten mit so mancher Software kommen. Im folgenden werden wir nur die drei Commodore-Laufwerke betrachten, die außerdem den Vorteil haben, daß man sie bei jedem Händler, der den C128 führt, erhalten kann.

1541 – der Oldtimer

Der Begriff »Oldtimer« trifft voll und ganz auf die 1541 zu. Denn zum einen hat dieses Laufwerk schon einige Jährchen auf dem Buckel (es ist seit 1982 erhältlich), zum anderen ist es wie ein Oldtimer auch recht langsam, vergleicht man es mit den neueren Modellen. Und ein letztes Kriterium trifft auf die 1541 wie auf Oldtimer zu: Sie ist mechanisch nicht sehr stabil, ja sogar »klapperig«. Das Wort »klapperig« ist übrigens wörtlich zu nehmen, denn beim Formatieren und bei Lesefehlern gibt die 1541 Geräusche von sich, die entfernt an einen Specht oder an Maschinengewehrfeuer erinnern. Für die technisch Interessierten zeigt Bild 1 die 1541 ohne den Gehäusedeckel.

Doch wollen wir hier die 1541 nicht zu sehr durch den Kakao ziehen. Denn für den C64-Besitzer ist sie das ideale Laufwerk: preiswert, einfach zu bedienen und relativ flexibel (was die zahlreichen Kopierschutzmethoden oder Schnelladesysteme beweisen). Wer allerdings Datenverwaltung oder Textverarbeitung machen möchte, der stößt bald an die Grenzen der 1541. Denn einerseits sind 170 KByte Speicherkapazität für viele Zwecke zu wenig, andererseits möchte man bei einer Datenverwaltung nicht dreißig Sekunden warten, bis man einen bestimmten Datensatz gelesen hat. Solche Wartezeiten sind bei der 1541 durchaus üblich. Gerade diesem Problem haben sich schon einige Hardware-Entwickler angenommen. Entstanden sind dabei Zusätze wie »SpeedDos«, »Turbo-Access« oder »Prologic Dos«, die die 1541 um Faktoren zwischen 3 und 25, je nach Art des Datenzugriffs, beschleunigen. Einziger Nachteil: Augenblicklich funktionieren diese Hardware-Zusätze nur im C64-Modus des C128, im C128-Modus und unter CP/M bleibt die 1541 langsam wie bisher. So wird mit der 1541 das Kopieren einer Diskette unter CP/M zur Geduldsprobe: Ganze 50 Minuten werden benötigt, um sich eine Sicherheitskopie der Systemdiskette anzulegen. Daß professionelles Arbeiten hier absolut unmöglich ist, dürfte jedem einleuchten.

Um noch ein paar Worte über die Stabilität der Laufwerksmechanik zu verlieren: Bei der 1541 wird der Schreib/Lese-Kopf mit einem Stepper-Motor über die Diskette bewegt. Der Tonkopf-Antrieb ist aber nur auf die Stepper-Motor-Achse lose aufgesteckt, so daß sich der Tonkopf relativ leicht verstellt. Hinzu kommt, daß sich die 1541 relativ stark aufheizt, und das Netzteil, Elektronik und Mechanik gemeinsam in einem Metallchassis hängen, so daß sich die Mechanik erwärmt und leicht ausdehnt. Auch



Unentbehrlich für den C128-Beitzer: die Floppy-Laufwerke.



für den C 128?

das kann zu verstellten Köpfen führen. Wenn also nach längerem Betrieb laufend Lesefehler auftreten, ist eine Tonkopfeinstellung fällig, die man meistens von einem Fachmann ausführen lassen wird, was zu erheblichen Kosten führen kann.

Eine Weiterentwicklung der 1541 – so kann man die 1570 sehen. Das Gegenteil, ein Rückschritt von der 1571, trifft allerdings auch zu. Denn die Entstehungsgeschichte der 1570 ist recht abenteuerlich. Als die 1571 serienreif war, konnte Commodore sie wegen diverser kleiner produktionstechnischer Probleme nicht sofort in großen Stückzahlen herstellen. Um den C 128-Besitzern aber ein angemessenes Laufwerk zur Verfügung zu stellen, wurde aus der 1541 und der 1571 ein Mittelding, die 1570, entwickelt. Als Basis diente die Mechanik der 1541 und die Platine der 1571. Dies ist in Bild 2 deutlich zu sehen, vergleicht man die Platine mit der der 1571 (Bild 3) und die Mechanik mit der der 1541 (Bild 4). Die mechanischen Probleme der 1541 wurden bei der 1570 aber weitgehend beseitigt. So konnte das Problem der verstellten Schreib/Lese-Köpfe aufgrund einer leicht geänderten Stepper-Motor-Konstruktion und einer Sicherheits-Lichtschranke, die einen Kopf-Anschlag bei Spur Null verhindert, eingeschränkt werden. Die thermischen Probleme sind allerdings geblieben. Außerdem steht dem Benutzer immer noch die Datenmenge von nur 170 KByte zur Verfügung.

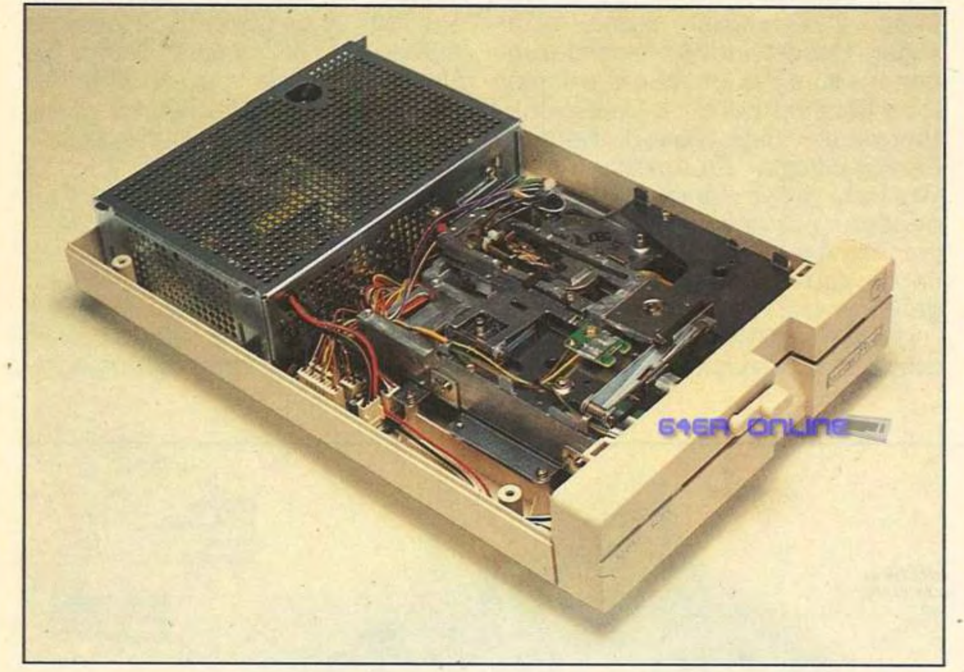


Bild 3. Die 1571 präsentiert sich mit stabiler Mechanik und guter Abschirmung.

1570 – die Verlegenheitslösung

Besonders interessant ist die 1570 aber wegen einiger zusätzlicher Bausteine auf der Platine. Mit denen ist es möglich, sogenannte MFM-Formate zu lesen und zu

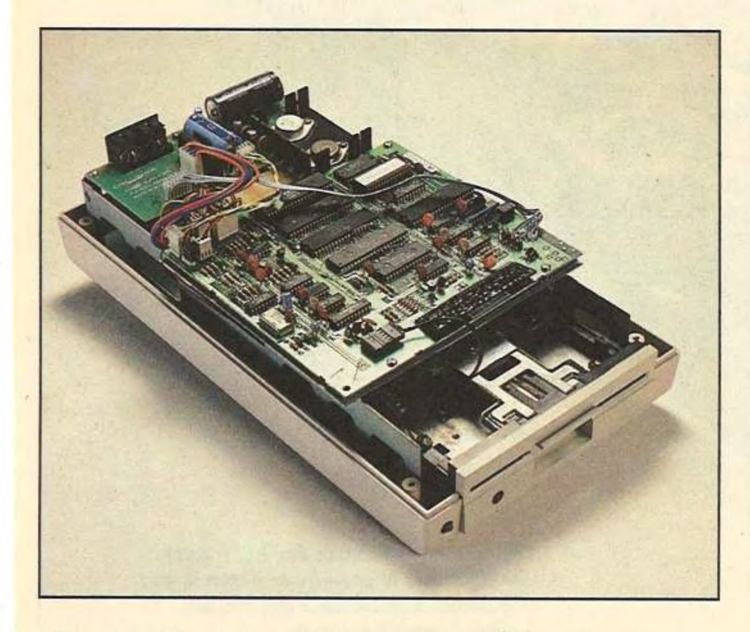


Bild 2. Die 1570 ist klar als Kreuzung zwischen 1571 und 1541 erkennbar.

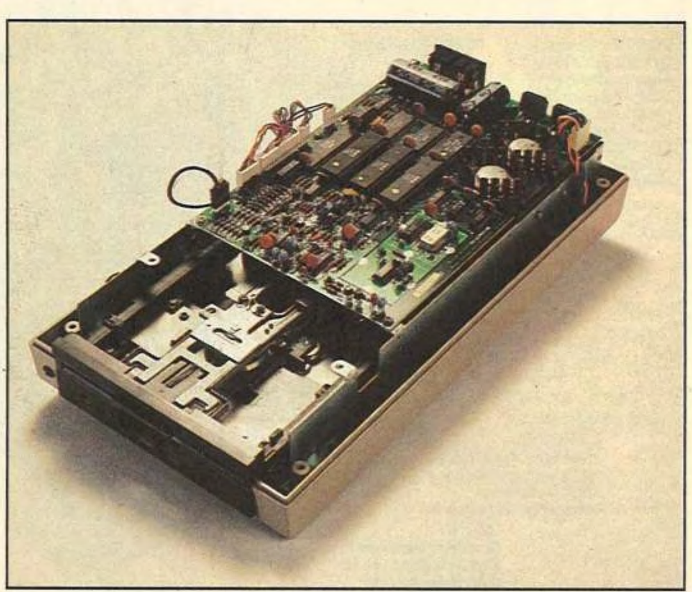


Bild 1. Die 1541 ist das älteste der drei vorgestellten Laufwerke. Bei den C64-Besitzern ist sie sehr verbreitet.

schreiben. Zur Erläuterung: Commodore verwendet bei allen Laufwerken das GCR-Format (Group Code Recording), um Daten auf der Diskette zu speichern. Die meisten CP/M-Rechner verwenden aber ein anderes Verfahren, das sich MFM oder Modified Frequency Modulation nennt. Die 1570 (wie auch die 1571) sollte nun aber für den CP/M-Betrieb beide Formate verarbeiten können, damit auch Disketten beispielsweise von einem Kaypro Personal Computer oder einem IBM einwandfrei gelesen und beschrieben werden können. Es befindet sich ein zweiter Floppycontroller-Chip auf der Platine, der eben für diese MFM-Formate verantwortlich ist. So sind auf der 1570 Disketten lesbar, die auf anderen Computern unter dem Betriebssystem CP/M beschrieben worden sind. (Für die PC-Besitzer sei erwähnt, daß keine MS-DOS-Disketten, sondern nur CP/M-86-Disketten weiterverarbeitet werden können.) Damit ist also der Datenaustausch zwischen unterschiedlichen Computern möglich. Es ist zum Beispiel eine Kleinigkeit, Texte, die man im Büro unter Wordstar auf einem Personal Computer unter dem Betriebssystem CP/M geschrieben hat, zu Hause mit dem C 128 weiterzubearbeiten.

Ein letzter, aber entscheidender Punkt ist die Geschwindigkeit des 1570-Laufwerks: Ein LOAD-Befehl wird zirka 5- bis 8mal schneller ausgeführt als mit einer 1541. Fünfmal schneller werden Programme geladen, die mit einer 1541 gespeichert wurden, achtmal oder noch schneller geht es, wenn mit der 1570 (oder 1571) gespeichert wurde. Dafür ist das Speichern selber nur zirka anderthalbmal schneller. Die Übertragung von sequentiellen und relativen Dateien wird um Faktoren zwischen zwei und drei beschleunigt. Beim Direktzugriff lassen sich sogar Geschwindigkeitssteigerungen bis zum Faktor zehn erreichen (zum Beispiel bei dem Datenbank-Programm »Superbase«). Andere, Floppy-interne Operationen wie »Scratch« oder »Validate« werden dagegen nur unwesentlich schneller. Flott geht auch noch das Formatieren, das viermal schneller ausgeführt wird.

Die 1571 schließlich ist das Flaggschiff der Commodore-Laufwerke für den C 128. Unser Bild 3 zeigt sie ebenfalls ohne den Gehäusedeckel. Alle Vorteile der 1570 gegenüber der 1541 gelten auch hier, es sind sogar einige neue hinzugekommen.

1571: das Flaggschiff

Da wäre zuerst einmal der wirklich stabile und professionelle Aufbau von Mechanik und Elektronik zu nennen. Das Netzteil ist komplett abgeschirmt und von der Mechanik wie der Elektronik vollkommen thermisch entkoppelt. Daraus folgt, daß es keine Schwierigkeiten mehr mit einer aufgeheizten Mechanik mehr geben wird. Ein weiteres großes Plus ist der erheblich bessere Aufbau der Mechanik. Damit Sie einen Eindruck vom Aufbau der 1571 bekommen, haben wir sie in Bild 4 für Sie auseinandergenommen.

Wesentlich an der 1571 ist aber ihr doppelter Schreib/Lese-Kopf. Damit hat die 1571 gleichzeitig auf beide Diskettenseiten Zugriff und kann damit auch die doppelte Datenmenge, 340 KByte, verwalten. Damit ist mit Datenverwaltungspro-

6522-VIA. I/O-Baustein für den Controller

6502-Prozessor, der der 1571 die »Intelligenz« gibt.

Die Schreib-/Lese-Mechanik. Im Gegensatz zur 1541 sind zwei Köpfe vorhanden, die eine Diskette beidseitig beschreiben und lesen können.

WD1770-Controller. Er wird mit einem entsprechenden Utility-Programm so programmiert, daß die Floppy-Station CP/M-Disketten von anderen Computern lesen kann.

251828-Controller

Port-Anschlüsse

DIP-Schalter. Hier kann die Floppy auf verschiedene Geräteadressen umgeschaltet werden.

6522-VIA. Programmierbarer Ein-/Ausgabe-Baustein für den seriellen Bus.

DOS-ROM, das Betriebssystem der 1571

2-KByte/8-Bit-RAM. Der »Floppyspeicher«, der auch von Ihnen genutzt werden kann.

6526-CIA. Dieser Baustein regelt, wie im C 64 und C 128 auch, die Speicherzugriffe.

Bild 4. Das Innenleben der 1571 unter der Lupe. Das abgeschirmte Netzteil, das sich normalerweise über der Platine befindet, wurde entfernt.



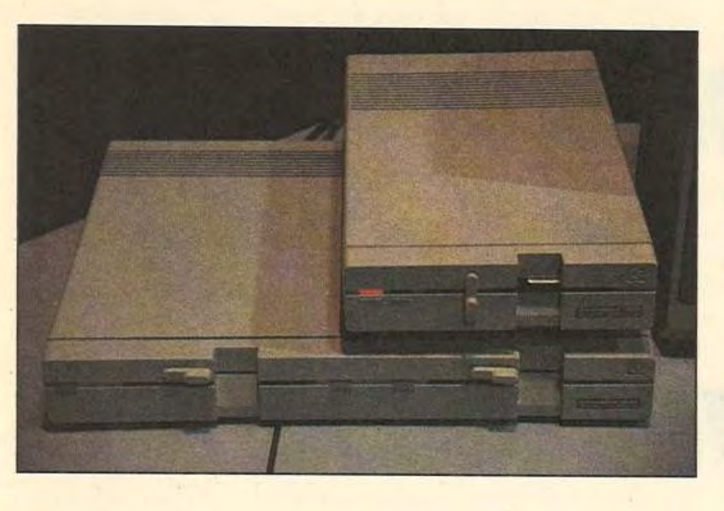


Bild 5. Noch nicht lieferbar ist das Doppellaufwerk 1572.

grammen schon relativ ordentliches Arbeiten auch in einem Kleinbetrieb möglich.

Viele Besitzer eines C 64 und einer 1541 sind nun in Sorge, ob ihre Programme, die ja teilweise mit recht »wilden« Kopierschutzverfahren ausgestattet sind, auf der 1571 mit dem C 128 noch funktionieren. Hier müssen leider einige kleine Abstriche gemacht werden. Programme, die nicht zu sehr das Floppy-DOS manipulieren, laufen noch einwandfrei, da so viele Einsprungadressen wie möglich beibehalten wurden. Beispiel hierfür sind »Hypra-Load« oder »Turbo-Nibbler«. Aber Ausnahmen bestätigen die Regel, so läuft beispielsweise das Programm »Quickcopy« nicht. Probleme bereiten dann noch einige der neuesten Kopierschutzmechanismen, da diese ganz genau überprüfen, ob eine Original-Commodore 1541 angeschlossen ist. Die 1571 kann diesem Test natürlich nicht standhalten. Allerdings teilten viele Softwarehersteller mit, ihren Kopierschutz bezüglich der Lauffähigkeit auf der 1571 zu überarbeiten. Aber wie gesagt ist es eigentlich schon eine Ausnahme, wenn einmal der Fall eintritt, daß ein Programm nicht auf der 1571 läuft. Das gleiche gilt für die 1570.

Welche Floppy für wen?

Nachdem Sie nun eine ganze Menge über die drei Laufwerke erfahren haben, fragen sich manche unter Ihnen sicherlich, welche denn nun die Richtige für ihn ist.

Die 1541 kann nur eingeschränkt empfohlen werden. Ihre Vorteile liegen in einem niedrigen Preis (zwischen 600 und 700 Mark) und vollständiger Kompatibilität zur 1541-Software, wie nicht anders zu erwarten. Für denjenigen der viel selber programmiert und weniger fertige Softwareprodukte wie Textverarbeitung oder Datenverwaltung verwenden will, ist sie eine preiswerte Alternative. Ebenso ist sie nützlich als Zweit-Laufwerk oder für denjenigen, der garantiert sämtliche C 64-Software laufen lassen will.

Die 1570 ist ein Zwischending, das niemanden vollkommen zufrieden stimmt. Nur wer unbedingt CP/M fahren will und auf jeden Pfennig (Preis zirka 800 Mark) achten muß, kann sich mit ihr anfreunden. Allerdings kommen viele CP/M-Programme mit den maximal 200 KByte Speicherplatz der 1570 nur mit Ach und Krach aus. Wer nur zwanzig KByte freien Platz bei einer Dateiverwaltung wie dBase hat, greift doch lieber bald auf einen Karteikasten zurück. Für nur zweihundert Mark mehr gibt es dann schon die 1571 (Preis zirka 1000 Mark), die doppelte Speicherkapazität und erheblich bessere Verarbeitung bietet. Wer mit CP/M arbeiten will oder auf Geschwindigkeit Wert legt, kommt ohne eines der beiden Laufwerke 1570/1571 nicht aus. Wer professionelle Programme unter CP/M laufen lassen will oder größere Datenmengen verwalten muß, hat bisher keine Alternative zur 1571 (außer zwei oder mehr 1571).

In USA angekündigt, jedoch noch nicht offiziell in Deutschland vorgestellt oder gar lieferbar ist das Doppellaufwerk 1572. Im Prinzip handelt es sich um zwei 1571 in einem Gehäuse, wie in Bild 5 erkennbar. Näheres über dieses Floppy-Laufwerk können Sie in der Zeitschrift 64'er lesen, sobald es auf dem Markt erscheint und wir es ausführlich testen können. Bis dahin müssen sich die C 128-Benutzer mit den Einzellaufwerken begnügen. (bs)

Fortsetzung von Seite 15

einem gleichzeitigen Bus-Zugriff von Prozessor und externem Gerät erweist sich der Prozessor meist als der Schwächere, was zu ernsthaften Problemen führen kann. Daß ein direkter Speicherzugriff ohne weiteres machbar ist, eröffnet dem C 128 gegenüber dem C 64 zusätzliche Einsatzgebiete in der Meßwerterfassung. Ein Meßgerät kann dadurch beispielsweise Meßwerte so schnell in den Speicher schreiben, daß eine Echtzeiterfassung eines Meßvorganges möglich ist. Eine andere Möglichkeit wäre der Anschluß eines Festplatten-Laufwerkes. Die Daten könnten damit viel schneller in den RAM-Bereich geladen oder aus dem Arbeitsspeicher geholt werden, wie wenn jedesmal der Prozessor jedes Bit vorher ein paarmal »umdreht«.

Der serielle Bus, Kassetten-Port und der User-Port des C 128 sind von den Anschlüssen her identisch mit denen des C 64. Die Bedienung des seriellen Bus wurde überarbeitet, so daß der C 128 zusammen mit dem neuen Commodore-Laufwerk 1571/1570 wesentlich schneller laden kann als der C 64 es kann. Im C 128-Modus lädt die 1571 sieben- bis achtmal so schnell wie die 1541. Ein HiRes-Bild mit 33 Blöcken benötigt nur noch ganze 3,2 Sekunden Ladezeit.

Schlußfolgerung

Der C 128 ist ein Zwitter. Bedingt durch die sicherlich löbliche Entscheidung von Commodore, den C 128 C 64-kompatibel zu machen ergaben sich bei der technischen Realisation natürlich einige umständliche Lösungen und Kompromisse. Die 8-Bit-Technologie ist bei Heimcomputern zwar noch weit verbreitet, dennoch zeigen einige Konkurrenten mit 16-Bit-Prozessoren, wo der Weg hingehen wird.

Auf der anderen Seite verfügt der C 128 bereits bei seiner Einführung über ein enormes Software-Potential (CP/M- und C 64-Software). Der Anwender kann also aus dem Vollen schöpfen. Das wesentliche Kaufkriterium wird aber der Preis sein. Wünschenswert wäre jeweils ein Kaufpreis von deutlich unter 1000 Mark für Computer, Floppylaufwerk und Monitor. Bei den bisherigen Preisen kommt man sehr schnell über 3000 Mark, und in dieser Preisklasse gibt es Alternativen.

(Bernhard Binz/hm)

Rushhour für CP/M auf dem 1541-Laufwerk

Der größte Nachteil des 1541-Laufwerks gegenüber der 1570/1571 ist die geringere Geschwindigkeit. Mit dem »Formel C«-Modul wird die 1541 ihren großen Brüdern angepaßt.

Der C 128 ist unter anderem ein CP/M-Computer. Das heißt, der Computer kann mit dem Betriebssystem CP/M Plus Version 3.0 arbeiten. Dafür verfügt die Hardware des C 128 über einen Z80-Prozessor. Dieser befindet sich neben dem neuentwickelten 8502-Prozessor auf der Platine des C 128. Erst mit ihm läuft das auf dem Z 80-Maschinencode aufgebaute Betriebssystem CP/M.

Im CP/M-Modus sind die Laufwerke 1541, 1570 oder 1571 verwendbar. Sie unterscheiden sich im wesentlichen im Speicherplatz den sie anbieten und - noch wichtiger - in ihrer Geschwindigkeit.

Neuer Glanz für die 1541

Hier sitzt allerdings auch der Haken für die 1541. Um in CP/M-Modus des 128 vernünftig zu arbeiten, ist die Geschwindigkeit der 1571/1570 notwendig.

Mit dem 1571-Laufwerk wird eine Datentransfergeschwindigkeit von 5200 Zeichen pro Sekunde erreicht. Die 1541 schafft nur ihre normale Übertragungsrate von 300 Baud. Das kommt hauptsächlich davon, daß bei der 1541 nicht alle Leitungen des Übertragungskabels vom Diskettenlaufwerk zum Computer genutzt werden. Die 1571 hat eine Leitung mehr für den Datentransfer zwischen Laufwerk und Computer zur Verfügung.

An diesem Punkt setzt das Formel C-Modul ein. Es erreicht mit der 1541 die Geschwindigkeit der 1571 indem es die Daten parallel überträgt und das Betriebssystem entsprechend abändert.

Flottes CP/M mit »Formel C«

»Formel C« besteht aus einem in den Expansions-Port einzusteckenden Modul und einer paralellen Datenleitung. Am Ende dieser Datenleitung ist ein Stecksockel, der in das Diskettenlaufwerk 1541 eingebaut wird. Über diese acht Kabel flitzen nun die Daten zirka 15mal schneller als normal hin und her. Diese Geschwindigkeit kann sogar mit einem weiter unten erklärten Trick auf das 20fache gesteigert werden. Selbstverständlich muß diese Datenübertragung entsprechend organisiert sein. Aus diesem Grund befindet sich in dem Modul ein EPROM, in dem die entsprechenden Routinen und Betriebssystem-Erweiterungen eingebrannt sind.

Ist das Modul installiert, wird nach dem Einschalten des C 128 sofort die Wirkung sichtbar. Das Betriebssystem CP/M wird auf der 1541 fast so schnell wie mit der 1571 geladen. Nach knapp 20 Sekunden ist CP/M betriebsbereit.

Zusätzlich weist das Steckmodul einige Besonderheiten auf, die recht nützlich sind. Es ist voll kompatibel zu CP/M und belegt keinen Speicherbereich des C 128. Somit wird dem C 128 speicherplatzmäßig keine Beschränkung auferlegt. Des weiteren ist eine neue Formatier-Routine eingebaut. Wenn eine Diskette von »Formel C« formatiert wurde, dann ist der Sektorabstand auf der Diskette kürzer als bei der normalen Formatierung. Dadurch wird der nächste Sektor beim Lesen oder Schreiben auch schneller gefunden und die Zugriffszeit verkürzt sich. Durch diesen Trick wird die Geschwindigkeit von »Formel C« bis auf das 20fache gesteigert. Diese Diskette kann trotzdem auch ohne »Fomel C« von jedem anderen Diskettenlaufwerk gelesen werden.

	1541	1541 mit »Formel C«	1570/71
CP/M laden	115 Sek.	24 Sek.	20 Sek.
Setup laden	20 Sek.	6 Sek.	5 Sek.
WordStar laden	80 Sek.	27 Sek.	25 Sek.

Zeitvergleichstabelle für CP/M

Schallgrenze erreicht

Die Geschwindigkeit könnte sogar noch mehr erhöht werden, aber die 20fache Geschwindigkeit stellt momentan für den CP/M-Modus des Commodore 128 die »Schallgrenze« dar.

Optional kann man sich für 49 Mark eine Centronics-Schnittstelle integrieren lassen. Diese Schnittstelle kann man aber auch nachträglich zum Modul nachrüsten. Dadurch wird der C 128 zu einem echten Bürocomputer mit allen notwendigen Anschlüssen.

»Formel C« bügelt die »Mängel« des 1541-Laufwerks gegenüber der 1570/1571 aus und gibt noch einige Bonbons dazu. Wer mit seiner »alten« 1541 - mit der Geschwindigkeit des 1570/1571-Laufwerks - am C 128 weiterarbeiten will, der kann sich das für 198 Mark verwirklichen. (zu)

Kein Bild ohne Monitor

Die Freude an einem Computer steigt und fällt mit dem Monitor. Was nutzt ein Supercomputer, wenn der Monitor nicht in der Lage ist, ein sauberes und scharfes Bild zu liefern?

Der C 128 ist ein Computer mit prinzipiell vier verschiedenen Video-Ausgängen, die unterschiedliche Normen aufweisen. Die Normen sind:

Pal, HF-moduliertes FBAS-Signal. Dieses Signal kann einen Fernseher über die Antennenbuchse ansteuern.

FBAS, das »gemischte« Farbsignal. Hier wird die Helligkeitsinformation zusammen mit der Farbinformation übertragen. Dieses Signal benötigt man für verschiedene Monitore, die keinen Composite-Anschluß haben und für Fernseher mit Video-Eingang. SCART (20-polige Buchse) oder VCR (8polige Buchse).

Composite. Bei dieser Norm wird die Helligkeits- und Farbinformation getrennt übertragen, wodurch man eine bessere Bildqualität erhält, als beim gemischten FBAS-Signal.

RGB. Die Farbinformationen werden in die Einzelfarben Rot, Grün und Blau getrennt und einzeln übertragen. RGB, das sind die Signale, die nach entsprechender Verstärkung die Bildröhre ansteuern. In jedem Farbfernseher werden in mehreren Stufen aus dem Hochfrequenz-PAL-Signal die RGB-Signale gewonnen. Die Bildqualitäten, die mit RGB-Signalen übertragen werden können, sind kaum zu übertreffen.

Doch warum hat der C 128 vier verschiedene Normen? Die Frage läßt sich mit der Kompatibilität des C 128 zum C 64 erklären. Der C 64 hat zur Erzeugung eines Bildes mit 40 Zeichen in 25 Zeilen drei Normen: PAL, FBAS und Composite. Das Composite- und das PAL-Signal können über die 8polige Videobuchse abgezapft werden, das PAL-Signal an der Antennenbuchse. Diese drei Signale werden aus dem Video-Chip des C 64 gewonnen, dem VIC 6567. Diesen Chip benötigte man im C 128, um einen C 64 hardwaremäßig nachvollziehen zu können. Man benutzt ihn auch im C 128-Modus, wenn eine 40-Zeichendarstellung gewünscht wird. Nur hat der VIC einen entscheidenden Nachteil, er liefert Composite-Signale, die keine vernünftige 80-Zeichendarstellung pro Zeile zulassen. Und gerade dies sollte der C 128 bei seinem Entwurf können. Zusätzlich sollte der C 128 wahlweise auch mit mit 2 MHz getaktet werden. Aus diesen beiden Gründen wurde ein neuer Chip für die Bilderzeugung nötig: ein Video-Controller (8563). Dieser Baustein erzeugt RGB-Signale zur Ansteuerung eines RGB-Monitors und ein Luminanzsignal für monochrome Monitore. Befindet sich der C 128 im 80-Zeichen-Modus, ist dieser Video-Controller aktiv. Aus diesem Grund braucht der C 128 im Prinzip zwei Farbmonitore, einen für die 40-Zeichen-Modi und einen für die 80-Zeichendarstellung.

Einen Ausweg aus dem »Zwei-Monitor-Dilemma« bietet der Commodore Monitor 1901. Er hat Eingänge für beide Normen, Composite und RGB. Mit einem Umschalter an der Frontseite wird zwischen den beiden Normen umgeschaltet. Dabei müssen keine Kabel umgestöpselt werden.

Im 64'er, Ausgabe 1/86 haben wir diesen Monitor getestet und waren einhellig der Meinung, daß der 1901, besonders bei 80 Zeichen pro Zeile, ein sehr scharfes Bild liefert. Diese gute Eigenschaft wird allerdings durch »Geisterbilder« getrübt. Ein helles Zeichen am Zeilenanfang und die ganze Zeile wird bei ungünstiger Kontrasteinstellung deutlich heller. Der Preis soll 1098 Mark betragen.

Ein anderer umschaltbarer Monitor, der für den C 128 geeignet ist, ist der Orion CCM-1280, der für unter 1000 Mark zu haben ist. Leider befindet sich der Umschalter auf der Geräterückseite und ist so nur schwer erreichbar. Beide Monitore haben ein Audioteil eingebaut.

Ein anderer Vertreter dieser Monitor-Klasse ist der Picom 4B/PR. Allerdings hat dieser Monitor kein Audioteil.

Außer diesen beiden Monitoren gibt es noch einige mehr, die sowohl RGB- als auch Composite-Signale »verstehen«. Allerdings nicht unbedingt die des C 128. Meist sind Synchronisationsprobleme dafür verantwortlich. Häufig hilft ein Invertieren der Sync-Impulse, was für einen Techniker leicht zu bewerkstelligen ist.

Fernseher, die über eine SCART-Buchse verfügen, können in der Regel auch an den C 64 angeschlossen werden. Denn eine SCART-Buchse läßt den Anschluß von RGB-Signalen zu. Die Bildqualität, die von einem Fernseher damit erreicht wird, kann sich sehen lassen. Zwar ist sie nicht so gut wie die eines Monitors, aber 80 Zeichen pro Zeile sind lesbar. Man sollte das jedoch nur als Notlösung ansehen.

Fernseher am C 128?

Zum Anschluß des C 128 an einen SCART-Eingang sollten Sie beachten, daß die Signalpegel des C 128 dafür zu hoch sind. Den Anschluß sollten Sie von einem Fernsehtechniker vornehmen lassen. Deshalb nur ein paar Hinweise. Die Spannungspegel der RGB- und Synchron-Signale des C 128 sollen etwa 5 V betragen. Wir haben 3 V gemessen. Die Spannung des Luminanzausgangs beträgt bis zu 3 V (gemessen 1,5 V bei weißem Bildschirm). Die RGB-Signale des Commodore 128 müssen über Spannungsteiler an die Empfindlichkeit des SCART-Eingangs angepaßt werden. Zur Synchronisation läßt sich das Luminanzsignal verwenden. Es muß an Pin 20 der SCART-Buchse angelegt werden. Man benötigt dann etwa 2 V am Pin für Austast-Blanking (Pin 16).

Ist Ihnen im Moment ein Farbmonitor für den C 128 zu teuer, sollten Sie sich fragen, ob ein monochromer Monitor nicht auch reicht. Dann müßten Sie einmal die Bauanleitung »Ein Monitor reicht« aus der 64'er, Ausgabe 10/85 (Seite 16), anschauen. Speziell für die C 128-Fans haben wir diesen Artikel in diesem Sonderheft nochmal abgedruckt. So haben Sie ein umfassendes Werk zum C 128.

Im folgenden finden Sie die Testergebnisse von fünf Monitoren für den C 128. Wir haben für Sie drei Farbmonitore gewählt, die sich problemlos an den C 128 anschließen lassen und zwei monochrome Monitore. Die beiden monochromen Monitore waren die Spitzengeräte in unserem Monitortest in Ausgabe 1/85.

Bevor Sie sich allerdings einen Monitor kaufen, sollten Sie genau abwägen, ob Sie sich einen monochromen Monitor oder einen Farbmonitor zulegen wollen. Verwenden Sie Ihren Computer als Schreibmaschine oder programmieren Sie beispielsweise naturwissenschaftliche Anwendungen, bei denen es auf Farbe nicht ankommt, dann sollten Sie sich einen monochromen Monitor kaufen. Sie tun Ihren Augen damit einen großen Gefallen. Sind Sie allerdings ein Spiele- oder Grafik-Fan, dann ist ein Farbmonitor für Sie das richtige Gerät. Beachten Sie auch, ob Sie einen Lautsprecher benötigen oder nicht.

Die Videobandbreite eines Monitors gibt die Leistungsfähigkeit des Videoverstärkers an. Die Videobandbreite eines Monitors kann man mit dem Frequenzumfang eines Audioverstärkers vergleichen. Der Pixelabstand gibt an, wie groß der Abstand der Leuchtpunkte in der Bildröhre ist. Je kleiner der Abstand ist (er wird in Millimeter gemessen), desto größer ist die Auflösung der Röhre. Je mehr solche Bildpunkte eine Röhre hat, desto größer muß die Bandbreite des Videoverstärkers sein.

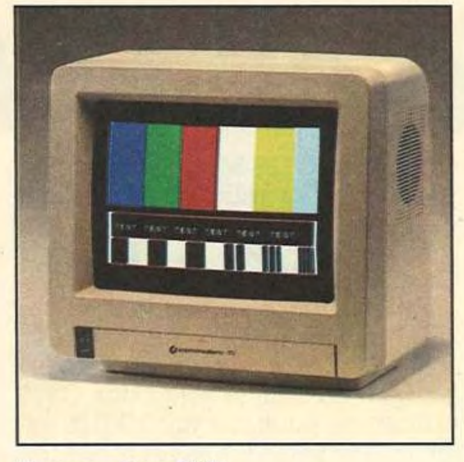
Sehr wichtig ist auch die Bildschirmdiagonale des Monitors. Für einen Heimcomputer kommen 9 bis 14 Zoll in Frage, wobei sich 12 Zoll häufig als das Optimum erweisen. Allgemein: Je näher Sie den Monitor vor Augen haben, desto kleiner sollte der Bildschirm sein. 5- oder 6-Zoll-Monitore sind für die tägliche Arbeit zu klein. (hm)

Unsere Testkriterien

Zum Test der Monitore entwarfen wir ein Testbild, das die Farbqualität und die Auflösung eines Monitors testet. Da die hier getesteten Monitore wahrscheinlich nur an Heimcomputer angeschlossen werden, die kein »Superbild« liefern (können), haben wir auf einen Test mit einem Farbbildgenerator verzichtet, denn hier würden eventuell Mängel sichtbar, die mit dem Bildsignal eines Heimcomputers »nie und nimmer« relevant werden.

Die monochromen Monitore testeten wir mit dem Luminanzausgang der RGB-Buchse des C 128.

Unser Testbild zeigt, wie gut der Monitor einen Farbwechsel und Schwarzweiß-Wechsel verkraften kann. Die vergrößerte Textdarstellung soll die Schärfe der Zeichendarstellung belegen.



Commodore 1901

Der Monitor zum C 128. Er »versteht« sowohl Composite- als auch RGB-Signale. Mit einem Schiebeschalter an der Frontseite wird zwischen beiden Normen umgeschaltet. Man fragt sich natürlich, warum man bei Commodore keine automatische Umschaltung vorgesehen hat, so daß man mit der 40/80-Zeichentaste auch die Monitoreingänge hätte umschalten können.

Die Bildschärfe des 1901 ist als sehr gut einzustufen, gemessen an den Mitbewerbern. Schwarzweiß-Wechsel werden sehr gut verkraftet. Bei einem Wechsel der Grundfarben treten zwischen den Farbbalken schwarze Linien auf.

Der RGB-Modus besticht durch eine exzellente Schärfe der Zeichen, obwohl der 1901 »nur« eine Lochmaske besitzt. Schlecht auf die Bewertung wirken sich die »Geisterbilder« aus. Ein helles Zeichen am Zeilenbeginn erhellt die ganze nachfolgende Zeile. Durch geeignete Helligkeits- und Kontrasteinstellung können die »Geisterbilder« jedoch so unterdrückt werden, daß sie kaum noch sichtbar sind.

Positiv
Alle Bedienungselemente an der Frontseite
Sehr guter Abgleich
Eingebautes Audioteil

Negativ
Geringer Kontrastregelbereich im Composite-Modus
»Geisterbilder«
Kein Anschlußkabel



Info: Commodore Büromaschinen, Lyoner Str. 38, 6000 Frankfurt, Tel. 069/6638-0, Preis: 1098 Mark



Orion CCM 1280

Auch dieser Monitor läßt sich ohne weiteres an den C 128 anschließen. Dazu wird ein Kabel mitgeliefert.

Zwischen dem Composite- und dem RGB-Modus wird mit einem Schalter umgeschaltet. Leider ist dieser Schalter an der Rückseite nur schwer zu erreichen.

Der Monitor kann auch zwischen RGB-TTL und RGB-Analog umgeschaltet werden. Beim Composite- und FBAS-Eingang heißt es aufpassen: »Separate« bedeutet Composite und »Composite« bedeutet FBAS. Außerdem sind die Farben der Chrominanz- und Luminanz-Buchse vertauscht.

Die Zeichenschärfe im 80-Zeichenmodus ist etwas geringer als die des 1901, trotz der Schlitzmaske.

Im Composite-Modus kann ein höherer Kontrast als beim 1901 eingestellt werden. Bei Schwarzweiß-Wechseln sind leichte Unschärfen an den Kanten festzustellen, die aber noch vernachlässigt werden können.

Positiv
Eingebautes Audioteil
FBAS-Eingang vorhanden

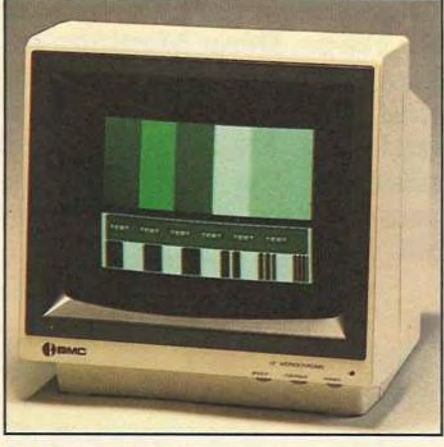
Negativ
RGB/Composite-Umschalter an der Rückseite



Bezugsquelle: Hard&Soft, Gagernstr. 4, 8580 Bayreuth, Tel. 0921/68877, Preis: 998 Mark









Picom 4B/PR

Der Picom hat einen FBAS- und einen RGB-TTL/Analog-Eingang. Damit kann sowohl der 40- als auch der 80-Zeichen-Modus des C128 genutzt werden.

Die Bildqualität bei RGB-Ansteuerung entspricht in etwa der des Orion CCM 1200.

Das Testbild zeigt, daß der Picom Farbwechsel sehr gut verkraftet. Nur zwischen Grün und Rot ist eine schwarze Kante vorhanden. Beim Linientest zeigt der Picom allerdings Schwächen: Weiße Linien auf schwarzem Grund werden farbig.

Die Textschärfe ist minimal besser als die des Sanyo CD3195C.

Die an der Rückseite angebrachten Regler zur Kontrast- und Farbeinstellung müssen mit einem Schraubendreher eingestellt werden. Das ist eine wesentliche Einschränkung des Bedienungskomforts.

Der Monitor hat kein Audioteil. Das bedeutet, daß man sich einen externen Audioverstärker beschaffen muß, will man nicht die heimische Stereoanlage ständig am Computer anschließen.

Positiv
Gutes Ergebnis im Farbbalkentest

Negativ
Kein Audioteil
Regler an der Rückseite nur mit Schraubendreher zugänglich



Mirwald Elektronik, Fasanenstr. 8, 8025 Unterhaching, Tel. 089/6111224, Preis: 1584 Mark

BMC BM-12G

Der BMC BM-12G konnte durch die beste 80-Zeichendarstellung überzeugen. Die Bildschirmfarbe ist, wie weithin üblich, Grün.

Für Grün ist das menschliche Auge am empfindlichsten. Das heißt, daß für das Auge Grün auf Schwarz einen größeren Kontrast bildet als Orange auf Schwarz. Viele Computerbesitzer geben deshalb Grünmonitoren den Vorzug.

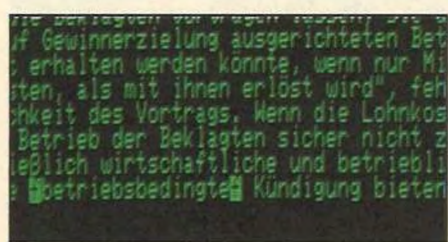
Wie bei monochromen Monitoren üblich, wird der BMC-Monitor über ein Luminanz(BAS)-Signal angesteuert.

In die Eingangsleitung kann über einen Schiebeschalter an der Rückseite ein 75-Ohm-Abschlußwiderstand geschaltet werden. Die Videobuchse des BM-12G ist »durchgeschleift«. Dadurch könnte beispielsweise ein Videorecorder zu Dokumentationszwecken an den Monitor angeschlossen werden.

Die Regler für Kontrast und Helligkeit befinden sich an der Frontseite.

Positiv
Sehr gute Textwiedergabe
Kontrast- und Helligkeitsregler an der Frontseite

Negativ
Kein Audioteil



Mirwald Elektronik, Fasanenstr. 8, 8025 Unterhaching, Tel. 089/6111224, Preis 379 DM

Taxan KX1201E

Schon seit längerer Zeit gibt es diesen Monitor. Die Qualität der Textdarstellung ist von der des BMC-12G kaum zu unterscheiden. Die etwas höhere Schärfe des BMC-12G läßt sich nur durch sehr genaues Hinschauen erkennen.

Wie der BMC-12G hat auch der Taxan KX1201 kein Audioteil. Also keinen Tonverstärker und keinen Lautsprecher.

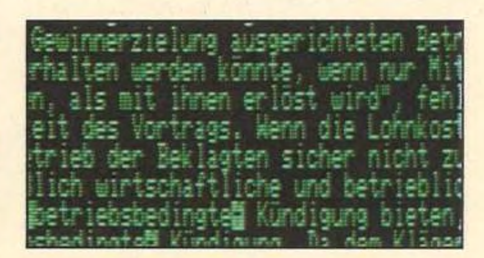
An der Frontseite befindet sich der Netzschalter und der Helligkeitsregler. Der Kontrast wird an der Rückseite eingestellt.

Der Taxan sollte mit dem Luminanzsignal des C64 angesteuert werden. Durch das FBAS-Signal wird der Bildschirm streifig.
Das gilt aber nicht nur für den Taxan, sondern für alle monochromen Monitore. Denn die dem Helligkeitssignal überlagerten Farbkomponenten verschlechtern die Bildqualität der S/W-Darstellung.

Der Taxan KX1201E läuft bei uns in der Redaktion seit mehr als einem Jahr ohne Störungen.

Positiv
Gute Bildwiedergabe

Negativ
Kontrastregler an der Rückseite
Kein Audioteil



Melchers & Co, Schlachte 39/40, 2800 Bremen 1, Tel. 04211/176989, Preis: 398 Mark

Kab

Paßt oder paßt nicht – das ist die Frage. Denn Monitore an den C 128 anzuschließen ist eine Wissenschaft für sich. Studieren Sie mit uns und erlangen Sie das »Summa cum laude« der Kabeltechnik.

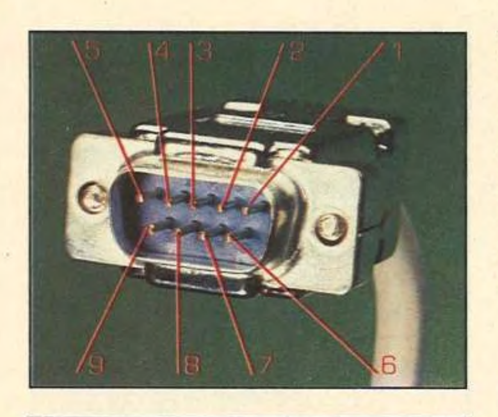
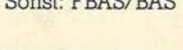


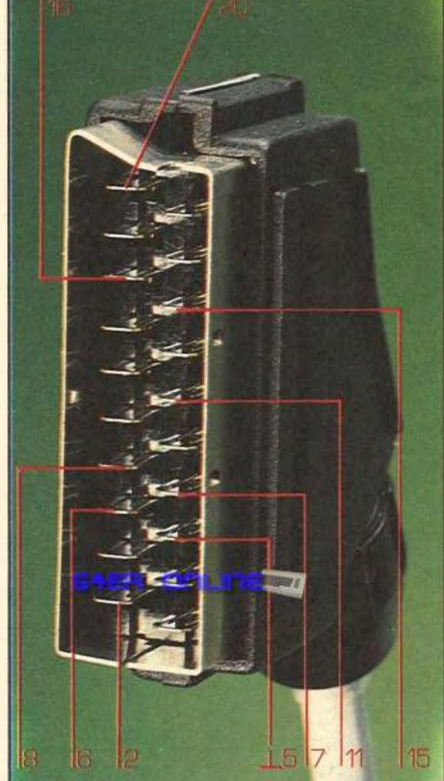
Bild 2. Cannon 9polig ▲

1, 2	Masse
3	R
4	G
5	B
6	Intensität
7	Luminanz (Monochrom-Signal)
8	Vertikale Synchronisation
9	Horizontale Synchronisation

Bild 8. SCART-Stecker ▶

2	Audio links
5,17	Masse
6	Audio rechts
7	Blau
8	Schaltspannung (12 V)
11	Grün
15	Rot
16	Austast-Blanking (1-3 V)
20	Bei RGB: Synchronisation Sonst: FBAS/BAS





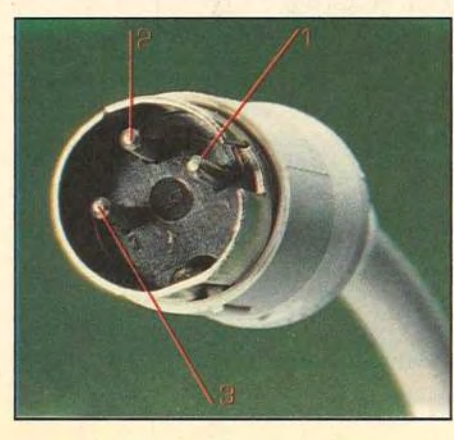
◀ Bild 5. Kleingerätestecker
7polig für C 64

1	Luminanz
2	Masse
3	Audio out
4	Video out
5	Audio in
6, 7	unbelegt



Bild 4. DIN 3polig ▶

1	FBAS
2	Masse
3	Audio



Der C 128 bietet wahrlich eine exzellente Schnittstellenvielfalt zum Anschluß eines Monitors. Einziger Haken an allem: Die Norm fehlt. Zwar sind (Achtung Insiderinformation) alle notwendigen Signale vorhanden, dafür fehlt aber für den Anschluß eines zufällig vorhandenen, oder möglicherweise preisgünstig erstandenen Monitors das passende Kabel. Aber was ist schon ein Kabel im Vergleich zu einem exzellenten, flimmerfreien Farbbild in allen Lagen? Lassen wir uns also nicht von den verwirrenden Signalen aus dem Konzept bringen und machen uns an die Herstellung der Verbindung. Zunächst ist es natürlich notwendig, die Belegungen aller Schnittstellen zu betrachten. Am C 128 findet man zwei Ports zum Anschluß eines Monitors. Der erste Anschluß ist sicherlich allen C 64 Besitzern bekannt – es ist der 8polige Kleingerätestecker, dessen Belegung Sie in Bild 1 finden. Neu dagegen ist der 9polige Cannon-Stecker, der zum Anschluß eines RGB-Monitors gedacht ist (Anschlußbelegung in Bild 2).

Wer für wen

Wer nun zu dem Schluß gekommen ist, daß es ja letztendlich gleichgültig ist, an welchem Port der Monitor angeschlossen wird, muß leider enttäuscht werden. Der 8polige Kleingerätestecker (wie beim C 64) eignet sich nur für 40 Zeichen und 25 Zeilen sowie der farbigen 320 x 200 Grafik. Andererseits ist es dem RGB-Port vorbehalten, 80 Zeichen darzustellen. Falls Sie keinen Monitor haben, der beide Anschlüsse besitzt, (umschaltbar) sollten Sie sich für eine der beiden Lösungen (Composite oder RGB) entscheiden, oder den später beschriebenen Trick anwenden. Das Kabelproblem ist deswegen natürlich noch nicht gelöst. Deshalb ist es sicherlich sinnvoll, die wichtigsten Möglichkeiten von Anschlüssen zu betrachten, mit denen Ihr Monitor ausgerüstet sein könnte.



elsalat

1. Anschlüsse von monochromen Monitoren

Bei den monochromen Monitoren mit FBAS-Eingang gibt es glücklicherweise so etwas wie einen Standard. Bei fast allen auf dem Markt verkauften Geräten findet man eine Cinch-Buchse. Die Anschlußbelegung eines Cinch-Steckers dafür finden Sie in Bild 3. Es kann auch vorkommen, daß Sie eine drei-, fünf-, sieben- oder achtpolige Kleingerätebuchse vorfinden. In diesen Fällen sollte Ihr Stecker auf Pin 1 (Bild 1 und Bild 4 bis 6) mit dem Luminanz-Signal (Helligkeit) beschaltet sein. Sie können natürlich auch das Video-Signal (FBAS) verwenden, haben dann allerdings eine leichte Qualitätseinbuße, da ja das Video-Signal auch das bei monochromen Monitoren nicht benötigte Chrominanz-Signal enthält. In seltenen Fällen kann es vorkommen, daß Ihr Monitor mit einer BNC-Buchse (BNC-Stecker zeigt Bild 7) ausgestattet ist. In diesem Fall sollten Sie das Luminanz-Signal auf den Mittelstift legen. Falls Ihr Monitor eine 9polige Cannon-Buchse hat (Bild 2), genügt es, das Luminanz-Signal an Pin 7 der Cannon-Buchse zu legen. Ganz gleich, welchen Stecker Sie verwenden – vergessen Sie auf keinen Fall die Masse (Anschlüsse siehe Bild 1 bis 7).

2. Anschlüsse von farbigen oder monochromen Fernsehern mit Monitor-Eingang

Viele Fernseher sind mit einem Monitor-Eingang (AV/SCART) ausgerüstet. Man hat dadurch den Vorteil, die Modulation-Demodulation des Video-Signals auf einen hochfrequenten Träger zu umgehen. Solche Monitor-Eingänge sind hauptsächlich zum Anschluß an Videorecorder vorgesehen. Man kann diese Buchse aber durchaus auch für den Computer verwenden. Wenn Sie an der Rückseite Ihres Fernsehers eine Cinch-, BNC- oder Kleingeräte-Buchse vorfinden, gilt das gleiche wie für monochrome Monitore (Luminanz- oder FBAS-Signal verwenden).

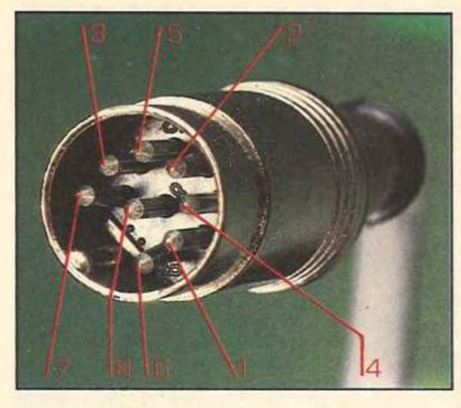
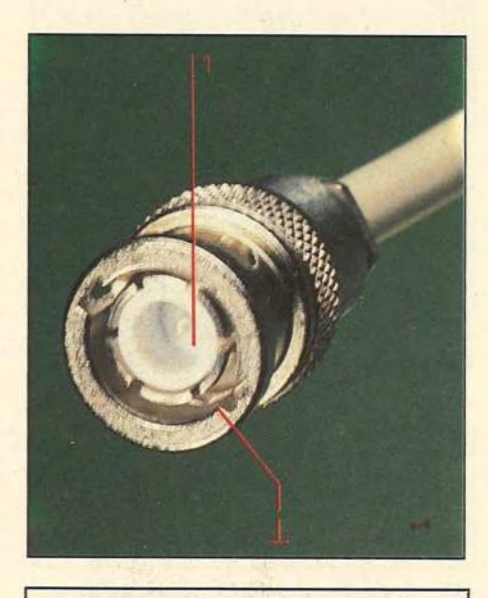


Bild 1. Kleingerätestecker 8polig für C 64 ▲

1	Luminanz
2	Masse
3	Audio out
4	Video out
5	Audio in
6, 7	unbelegt
8	Chrominanz



◄ Bild 7. BNC-Stecker

1 FBAS Masse

Bild 9. DIN 5polig (VCR-Norm) ►

1	Schaltspannung für Wiedergabe
2	Video in
3	Masse
4	Audio in
5	+12 V



Bild 6. DIN 5polig ►
am alten C 64

1	Luminanz
2	Masse
3	Audio out
4	Video out
5	Audio in

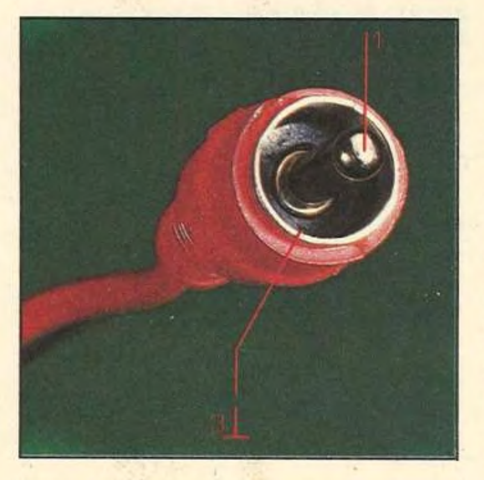


Bild 3. Cinch-Stecker ▲

1 FBAS 3 Masse

Etwas komplizierter wird das Ganze, wenn Ihr Fernseher mit einem SCART-Stecker ausgestattet ist. Die in Bild 8 dargestellte Anschlußbelegung zeigt die gebräuchlichsten Signale. Hier reicht es, wenn Sie das Luminanz- oder Videosignal an Pin 20 anschließen. Eine Besonderheit stellt die 5polige Kleingerätebuchse der VCR-Norm (Bild 9). Bei ihr muß das Luminanz- oder Videosignal an Pin 2 angeschlossen werden, Masse befindet sich an Pin 3. Achten Sie darauf, daß weder Pin 1 noch Pin 5 mit Ihrem Computer verbunden sind. Die dort vorhandene

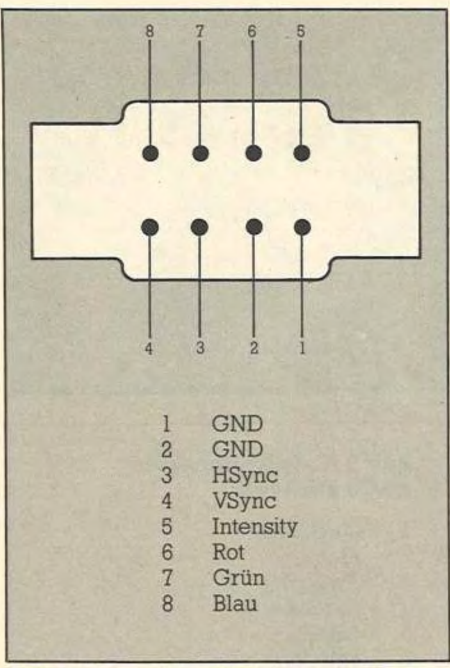


Bild 10. Der 8polige VTR-Stecker findet oft Verwendung

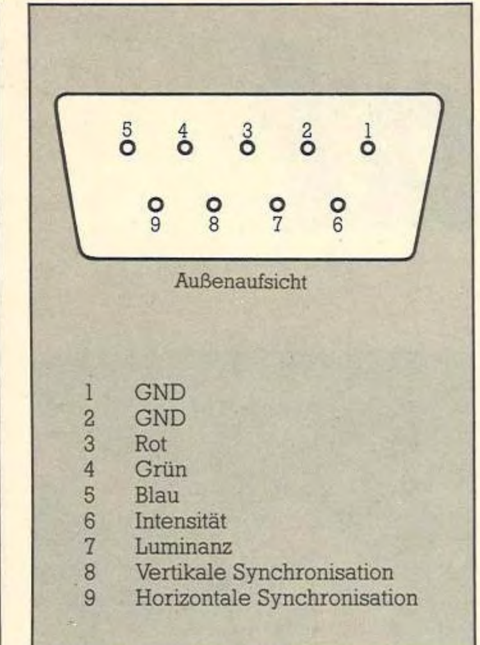


Bild 11. Der Cannon-Stecker des C 128

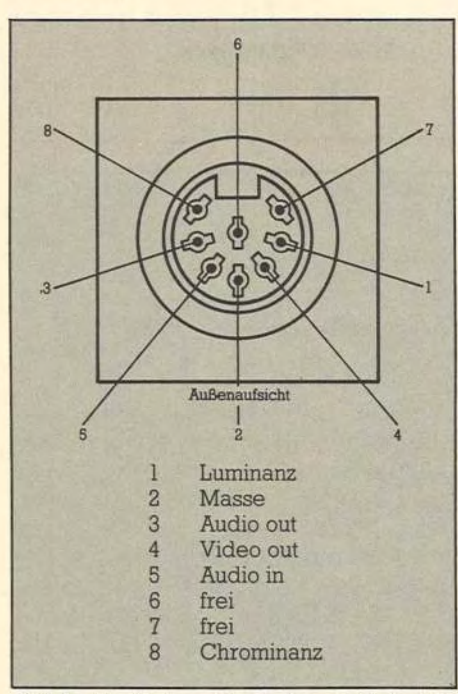


Bild 12. Die Kleingeräte-Buchse wie beim C 64

Cannon 9 polig	Scart	DIN 5	Cinch	BNC
Pin 7 mit	Pin 20	Pin 2	Pin 1	Pin 1
Pin 1 mit	Pin 5	Pin 3	Pin 2	Pin 2

Bild 13. So wird das 80-Zeichensignal an verschiedene Stecker gelegt

C 128	Signal	Centronics
A	GND	16
B	Flag = AKG	10
C	D0	2
D	D1	3
E	D2	4
F	D3	5
H	D4	6
J	D5	7
K	D6	8
L	D7	9
M	PAZ = Strobe	1

Bild 15. Anschlußschema für ein preiswertes Drucker-Kabel

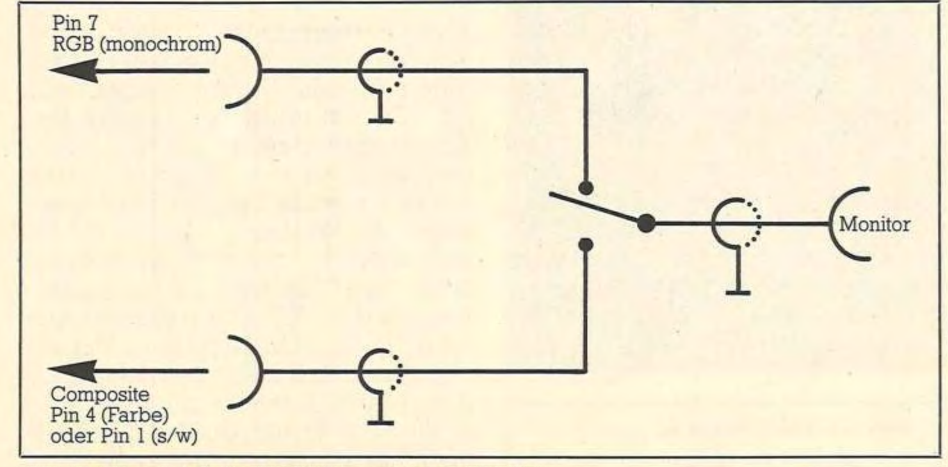


Bild 14. Dieser einfache Schaltplan spant einen Monitor

Schaltspannung zum automatischen Umschalten könnte für Ihren C 128 tödlich sein. Benutzen Sie deshalb statt der Schaltspannung lieber die AV-Taste Ihres Fernsehgerätes, wenn Sie den Fernseher als Videomonitor verwenden wollen.

3. Verwendung des RGB-Signals für RGB-Farbmonitore

Der C 128 liefert ein sogenanntes digitales RGB-Signal. Mit diesem digitalen Signal lassen sich beim C 128 acht Farben erzeugen.

Achten Sie deshalb darauf, daß Ihr RGB-Monitor ebenfalls ein digitaler Monitor ist. Solche Monitore sind in der Regel mit einem 9poligen Cannon- oder einem 8poligen VTR-Stecker (Bild 2 und 10) ausgerüstet. Bei diesen Anschlüssen genügt es, alle gleichnamigen Signale miteinander zu verbinden. Eine Besonderheit ist das beim C 128 an Pin 7 anliegende Luminanzsignal des RGB-Ausgangs. Dieses Signal läßt sich bestens dazu verwenden, einen monochromen Monitor mit BAS-Eingang anzuschließen. Auf einem monochromen Monitor werden bei Verwendung dieses Signals sogar 80 Zeichen und 25 Zeilen sehr scharf abgebildet (Anschlußschema: siehe Bild 11 bis 13)

Der Trick mit dem Schalter

Wem die Schaltung im Artikel »Ein Monitor ist genug« zu aufwendig ist, der kann sich natürlich auch mit einem einfachen Umschalter behelfen. Wie in Bild 15 zu sehen, wird zwischen dem RGB-Luminanzsignal und dem normalen Luminanzsignal (monochrom) oder dem Videosignal (Farbe) umgeschaltet.

Billige Centronics-Schnittstelle

Der C 128 besitzt, ebenso wie der C 64, einen User-Port. Und auch wie beim C 64 kann man diesen User-Port als Centronics-Schnittstelle umprogrammieren. Dies ist besonders für die Programme WordStar, Multiplan und dBase II, die es für den CP/M-Modus gibt, interessant, zumal die notwendige Schnittstellensoftware gleich eingebaut ist. Das Kabel kann sich jeder auf einfache Weise selbst herstellen. Alles was man braucht, ist ein User-Port-Stecker und ein Centronics-Stecker (Amphenol), sowie zirka einen Meter 16poliges Rund- oder Flachkabel. Bild 14 zeigt, wie die beiden Stecker miteinander verlötet werden müssen.

(aw)



Ein Monitor ist genug

Der C 128 benötigt eigentlich zwei Monitore: einen mit RGB- und einen mit Composite-Eingang. Mit dieser Bauanleitung für eine automatische Signalumschaltung reicht allerdings ein Monitor aus, wenn Sie im 80-Zeichenmodus auf Farbe verzichten können.

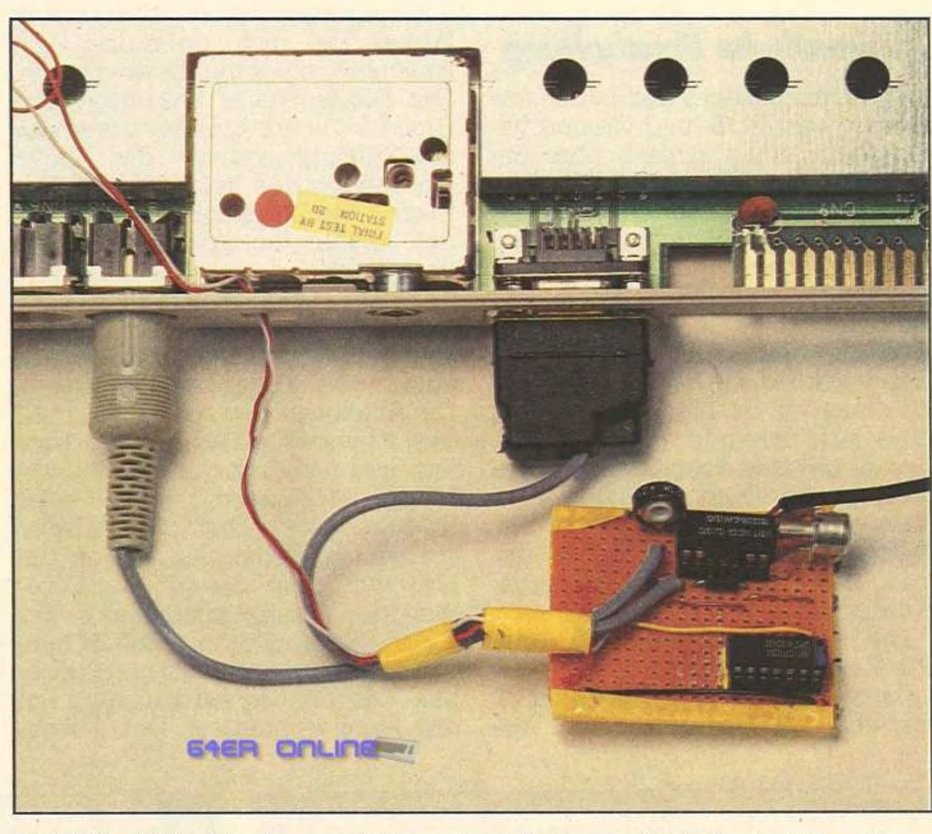


Bild 2. Die elektronische Normumschaltung. Zum Aufbau wurde eine Lochrasterplatine mit Streifen verwendet.

Kaum hatten wir die ersten C 128 in der Redaktion stehen, ärgerten wir uns über das lästige Umstecken des SW-Monitors. Composite-Ausgang im 40-Zeichenmodus, RGB im 80-Zeichenmodus. »So nicht!«, dachten wir und überlegten uns eine Schaltung, die Ihnen und uns in Zukunft die ewige Stöpselei erspart.

Schaut man sich die Belegung der beiden Video-Ausgänge des C 128 genauer an, fällt auf, daß beide Buchsen einen Luminanz (Helligkeits)-Ausgang haben. Beim RGB-Ausgang wird das Luminanz (Helligkeits)-Signal im Handbuch nur als Monochromsignal bezeichnet.

Mit einem Luminanzsignal kann jeder gebräuchliche SW-Monitor angesteuert werden. Auch der 1701/1702-Monitor von Commodore hat einen Luminanzeingang. Bei den letzteren kann auch der FBAS-Eingang an der Frontseite mit einem Luminanzsignal beschaltet werden. Man muß dann nur den Farbregler auf Schwarz-Weiß drehen.

Die einfachste Methode, ein und denselben Monitor sowohl an den RGB- als auch an den Composite-Ausgang anzuschließen, ist die Verwendung eines Adaptersteckers. Dazu wird an den RGB-Ausgang ein kurzes Zwischenkabel mit RGB-Stecker und Composite-Buchse angeschlossen. Bei der 80-Zeichendarstellung muß der Monitor an das Zwischenkabel angeschlossen werden, bei 40-Zeichendarstellung und C 64-Modus an den Composite-Ausgang des C 128.

Das bringt allerdings einige Probleme mit sich. Erstens wird das Umstecken schnell lästig und zweitens ist der Kontaktverschleiß an den Steckern und Buchsen sehr hoch. Es muß also eine automatische, verschleißfreie Umschaltung her.

Automatische Umschaltung, aber wie? Über dieses Problem haben wir uns Gedanken gemacht. Wir wollten Ihnen eine optimale Lösung anbieten, die einfach nachgebaut werden kann. Nach einigen Ideen wie User-Port-Schaltungen oder Logik-Bausteinen an der MMU und den beiden Videoteilen kam uns der richtige Einfall: die 40/80-Taste. Diese Taste, die direkt auf den Betriebsmodus Einfluß nimmt, müßte doch für die Umschaltung verwendbar sein. Tatsächlich wirkt diese Taste, ein 1 x EIN-Schalter (kein Taster), auf den Pin 48 der MMU. Dies ist der Eingang, über den die MMU den 40- oder 80-Zeichen-Modus nach dem Einschalten oder nach einem Reset initialisiert. Im Grundzustand hat dieser Eingang logischen High-Pegel.

Ist der 40/80-Zeichen-Eingang der MMU unbeschaltet, wird beim Initialisieren der 40-Zeichen-Modus aktiviert, andernfalls der 80-Zeichen-Modus. Die 40/80-Taste schaltet den MMU-Eingang auf Masse, wenn der 80-Zeichen-Modus aktiviert werden soll. Man braucht also nur die Leitung vom Schalter zur MMU anzapfen, und schon ist man anhand des Logikpegels über den Darstellungsmodus nach dem Einschalten oder einem Reset informiert. 0V bedeutet 80-Zeichenmodus, +5V zeigt die 40-Zeichendarstellung an. Da die Anschlüsse des 40/80-Schalters aus

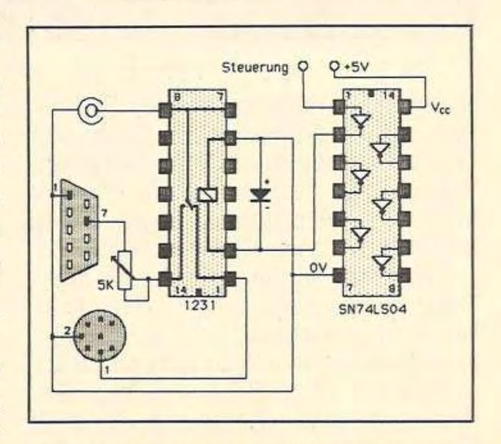


Bild 1. Der Schaltplan. Bei den Steckern werden die Lötseiten gezeigt.

der Grundplatte herausragen, ist das »Anzapfen« ein leichtes: Das Schaltkabel für die Umschaltelektronik muß nur an den Pin des 40/80-Zeichenschalters angelötet werden, der der C 128-Rückseite zugewandt ist.

Automatische Umschaltung

Die Umschaltung der Luminanzsignale vom RGB- und Composite-Ausgang erfolgt einfach über ein kleines Reed-Relais (1xUM) in einem DIL-Gehäuse. Diese Relais brauchen bei 5V Schaltspannung einen Schaltstrom von etwa 10 bis 20 mA. Zuviel für eine direkte Ansteuerung mit dem MMU-Eingang. Der Pegel vom MMU-Eingang muß also verstärkt werden. Ein TTL-LS-Hex-Inverter ist billig und besitzt eine ausreichende »Verstärkung«. Der Low-Power-Schottky-Typ sollte nicht durch einen normalen TTL-Baustein ersetzt werden, da der mehr Versorgungsstrom benötigt und einen kleineren Eingangswiderstand besitzt.

Die Umschaltplatine wird über Pin 2 des Kassetten-Ports mit +5V versorgt (siehe Handbuch). Es sind also nur zwei Drähte anzulöten. Verwenden Sie dazu am besten Schaltlitze, die knickfester als Draht ist. Der Lötkolben sollte eine Leistung von 16 Watt haben und gut vorgeheizt sein. Als Lötzinn eignet sich nur sogenanntes Elektroniklot.

Die Schaltung umfaßt nur wenige Teile (Schaltplan, Bild 1). Der Aufbau sollte deshalb nicht zu schwer sein. Haben Sie die Schaltung fertig aufgebaut (Bild 2,3,4), sollten Sie diese vor dem Anschluß an den Computer mit einer 4,5-Volt-Batterie überprüfen. Schließen Sie dazu Masse an den Minuspol der Batterie und +5V an den Pluspol der Batterie an. Wenn Sie nun die Steuerleitung an den Minuspol der Batterie legen, sollten Sie ein leises Klicken des DIL-Relais hören. Haben Sie nur einen Ersatztyp des angegebenen Relais bekommen, lassen Sie sich unbedingt die Anschlußbelegung davon zeigen. Der Diodentyp ist unkritisch. Die Diode dient nur zum Abfangen der Induktionsspannung, die beim Abschalten des Relais auftritt. Die Polung der Induktionsspannung ist der angelegten Spannung entgegengesetzt.

Ist die Schaltung soweit in Ordnung, können Sie die Stecker anschließen. Verwenden Sie dazu einadriges, abgeschirmtes Kabel. Die Abschirmung wird nur mit Pin 2 des Composite-Steckers verbunden und nicht mit dem Steckergehäuse, entsprechend mit Pin 1 des RGB-Steckers. Die Steckerbelegungen im Schaltplan zeigen die Lötseiten. Hier die genaue Belegung der Videobuchsen des C 128:

Für den Monitoranschluß ist eine Cinch-Buchse zum Aufbau vorgesehen. Die Buchse wird einfach auf der Platine angelötet. Der Außenkontakt wird mit Masse verbunden, der Innenleiter mit dem Signal.

Vor der Inbetriebnahme sollten Sie noch einmal alle Anschlüsse genau überprüfen. Ist alles in Ordnung, schließen Sie das Steuerkabel und die Spannungsversorgung an den C 128 an. Verbinden Sie den SW-Monitor mit der Cinchbuchse und die beiden Videostecker mit dem C 128. Dann können Sie den Computer endlich einschalten. Wenn Sie alles richtig angeschlossen haben, erscheint die Einschaltmeldung auf dem Bildschirm. Entweder im 40- oder 80-Zeichenmodus, je nachdem, wie die 40/80-Taste geschaltet ist. Erscheint im 80-Zeichenmodus kein Bild, drehen Sie das Trimmpoti und den Helligkeitsregler des Monitors voll auf.

Der Trimmer dient zur Abschwächung des Luminanzsignals vom RGB-Ausgang, da dieses Signal stärker als das des Chrominanz-Ausgangs ist. Der Abgleich ist sehr einfach:

1. C 128 einschalten
2. Mit ESC X den 40-Zeichenmodus einschalten (40/80-Taste entrasten)
3. Am Monitor Helligkeit und Kontrast für 40-Zeichen-Modus einstellen

RGB-Ausgang

Pin	Signal	Pegel	Impedanz
1	Masse	0V	–
2	Masse	0V	–
3	Rot	0/5V	75 Ohm
4	Grün	0/5V	75 Ohm
5	Blau	0/5V	75 Ohm
6	Intensität	0/5V	75 Ohm
7	Luminanz	0-3V_{SS}	75 Ohm
8	Horiz. Synch.	0/5V	75 Ohm
9	Vert. Synch.	0/5V	75 Ohm

Composite-Ausgang

Pin	Signal	Pegel	Impedanz
1	Lumin./Synch.	1V_{SS}	75 Ohm
2	Masse	0V	–
3	Audio-Ausg.	1V_{SS}	–
4	Composite	1V_{SS}	75 Ohm
5	Audio-Eing.		–
6	Chrominanz	1V_{SS}	75 Ohm

Belegung der RGB- und Composite-Buchse

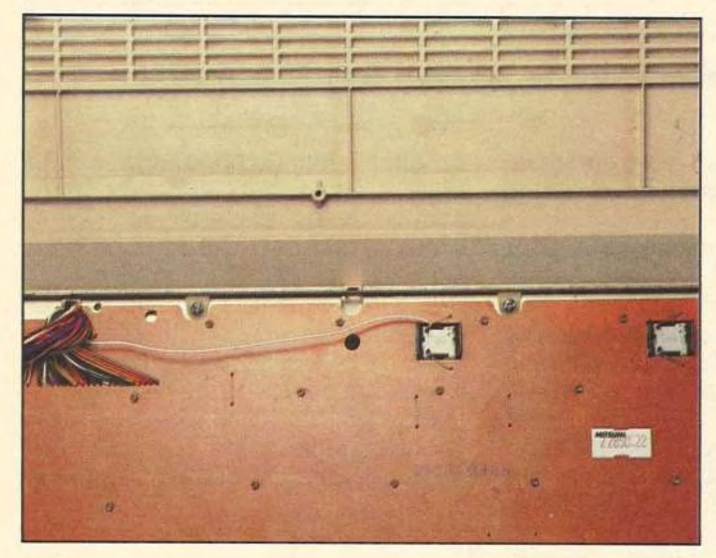


Bild 3. So ist die 40/80-Taste »anzuzapfen«

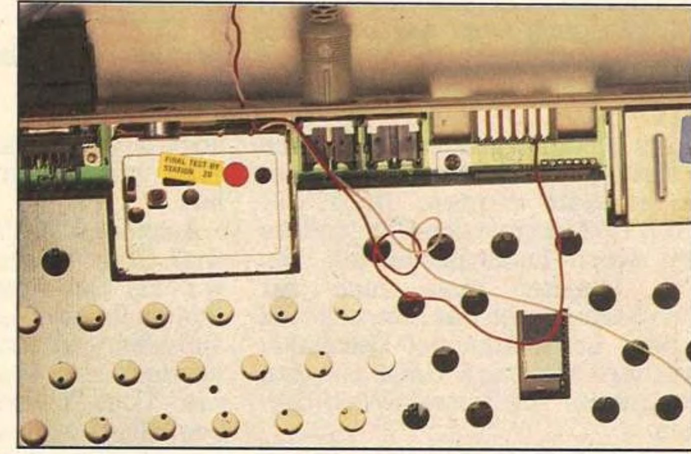


Bild 4. Die 5-V-Versorgungsspannung kann dem Kassetten-Port entnommen werden



1 74LS04

1 DIL-Reed-Relais 1xUM, Typ 1231 (Fa. Günther)

1 Trimmpoti 5 KOhm

1 Diode 4002 oä.

2 IC-Fassungen 14polig

1 Cinch-Platinen-Buchse

1 Kleingeräte-Stecker, 8polig, 270 Grad

1 Cannon-Stecker, 9polig mit Gehäuse

1 Stück Lochrasterplatine mit Kupferstreifen, etwas Schaltdraht und Litze

1 m einadriges abgeschirmtes Kabel

1 Gehäuse

Kosten: etwa 17 bis 20 Mark ohne Gehäuse

Stückliste zur Umschaltplatine

4. Mit ESC X 80-Zeichenmodus einschalten und 40/80-Taste drücken
5. Mit dem Trimmpoti die Helligkeit auf den 40-Zeichenmodus anpassen.

In der gleichen Weise können Sie die Helligkeit des 80-Zeichenmodus an die des C 64-Modus anpassen. Sie müssen nur immer die 40/80-Zeichentaste umschalten, wenn softwaremäßig zwischen 80 und 40 Zeichen pro Zeile umgeschaltet wird.

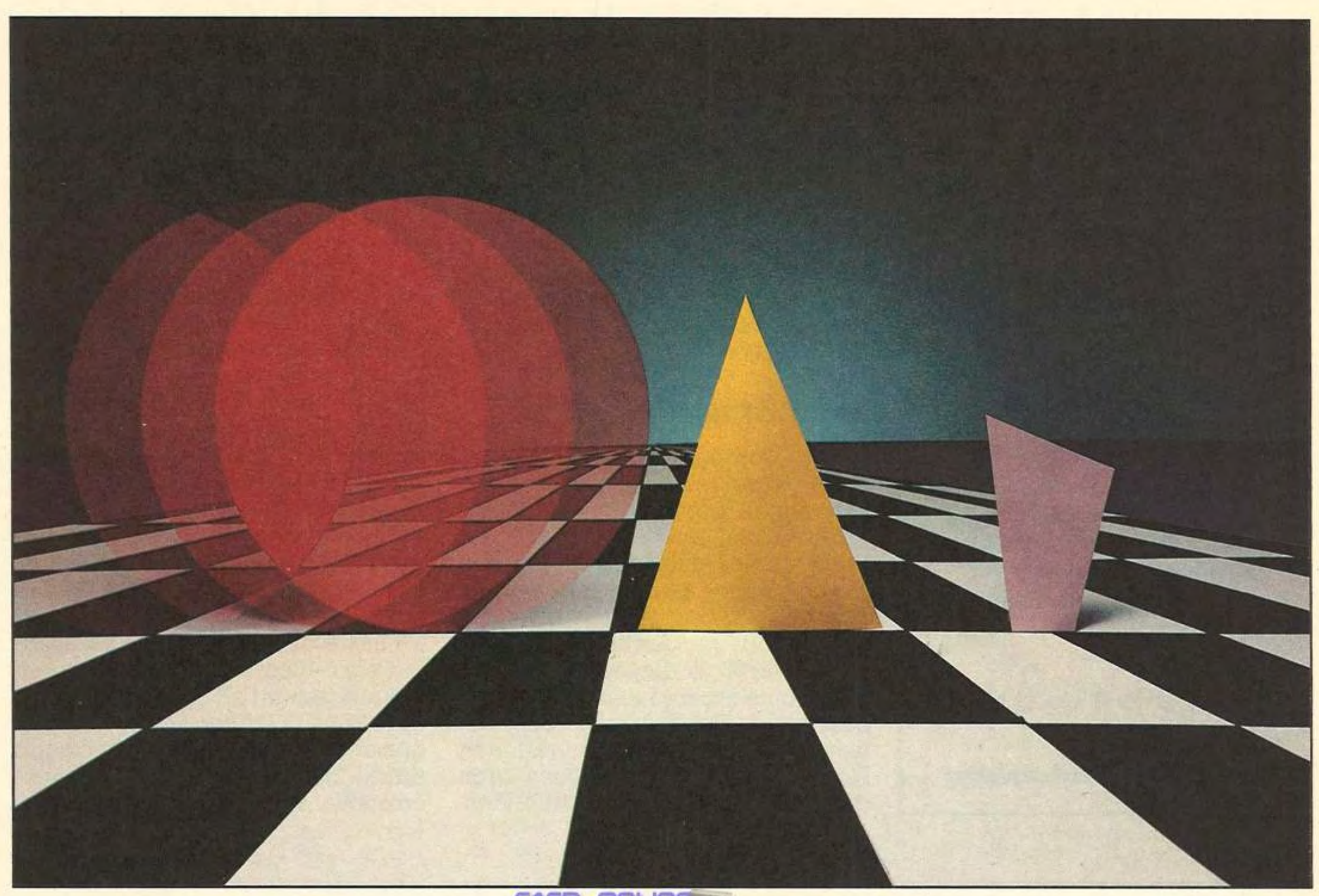
Farbe ist auch möglich!

Der Composite-Ausgang bietet neben dem Luminanz- und Chrominanzsignal noch ein komplettes Video-Signal an (FBAS, gemischtes Farb- und Helligkeitssignal). Mit diesem Signal kann jeder Farbmonitor mit Videoeingang angesteuert werden. Beim Commodore-Monitor 1701/1702 ist dieser Eingang an der Frontseite und kann mit einem Schalter an der Rückseite aktiviert werden. An diesen Eingang können Sie beim 1701/1702 übrigens auch das Luminanzsignal des RGB-Ausgang legen. Eventuelle Farbverschiebungen lassen sich beseitigen, indem man den Farbregler einfach auf SW dreht.

Um das FBAS-Signal auszunutzen, schließen Sie den automatischen Umschalter nicht an Pin 1 der Composite-Buchse an, sondern an Pin 4 (unterhalb Pin 1). Wenn Sie dann den Fronteingang (Einschalten!) des 1701/1702-Monitors mit der Cinchbuchse verbinden, erfolgt die 40-Zeichendarstellung (C 128, C 64) in Farbe. Der 80-Zeichenmodus bleibt Schwarz-Weiß. Der Nachteil dieser Lösung liegt in der schlechten Auflösung des 1701/1702. 80 Zeichen pro Zeile sind kaum noch zu entziffern.

Für die meisten SW-Monitore ist das FBAS-Signal nicht geeignet. Häufig stören dann Bildstreifen die Lesbarkeit.

Für welche der beiden Lösungen Sie sich auch entscheiden, bauen Sie auf jeden Fall die Schaltung in ein kleines Gehäuse ein. Nur so ist gesichert, daß kein Kurzschluß durch herumliegende Metallteile entsteht. (hm)



Sprites und Shapes auf dem C 128

Grafik und C 128: Das heißt optimales Zusammenspiel von Computer und Anwendung. Wie einfach Shapes und Sprites in der Handhabung sind, werden wir Ihnen auf den nächsten Seiten zeigen.

Um die Programmierung der Sprites auf dem C 64 zu erklären, ist schon viel Druckerschwärze verbraucht worden. Fast schien es eher die Aufgabe von Assembler-Programmierern zu sein, durch die POKE-Wüsten und Zahlensysteme zu gelangen um schließlich das Ziel – ein bewegtes Sprite auf dem Bildschirm – zu erreichen. Im C 128-Modus erspart uns das Basic 7.0 all diese Strapazen: Sprites zu erstellen und zu verwalten wird zum Vergnügen. Nachfolgend lernen wir die dazu nötigen Befehle bis in ihre Feinheiten kennen. Außerdem wird uns eine neue Gruppe von Grafikobjekten beschäftigen: die Shapes.

Sprites

Sollten Sie Ihnen noch nicht begegnet sein, die kleinen Kobolde (»sprite« ist englisch und heißt auf deutsch »Kobold« oder »Gespenst«), dann seien sie hiermit kurz vorgestellt. Es handelt sich um bewegliche grafische Objekte (daher auch der Ausdruck MOB = movable objects, der manchmal verwendet wird), die sowohl als mehr- als auch als einfarbige (dann hochaufgelöste) Figuren ohne Rücksicht auf den gerade eingeschalteten Grafikmodus munter über Text- und Grafikbildschirm huschen können. Die gesamte Verwaltung der Sprites geschieht durch den VIC-Chip, was ihr Auftreten auf den 40-Zeichen-Bildschirm beschränkt. Bevor Sie die folgenden Beispielprogramme ausprobieren, sollten Sie Listing 1 »Alle Sprites« abgetippt und gestartet haben.

Die Befehle zur Spriteprogrammierung

Zehn Basic-Befehle dienen zum Erstellen und Verwalten von Sprites:

SPRDEF

Damit ruft man den Sprite-Editor auf. Der Bildschirm wird gelöscht, links oben erscheint ein Raster mit 24 mal 21 möglichen Positionen, darunter fragt der Computer nach der gewünschten Sprite-Nummer.



Wir haben nun die Wahl zwischen acht vorgesehenen Sprites. Sollten Sie schon ein MOB im Speicher haben, dann rufen Sie es nun mit seiner Nummer auf, ansonsten müssen Sie sich jetzt festlegen: Alle weiteren Sprite-Befehle beziehen sich immer auf diese Nummer. Nach deren Eingabe wird das Rasterfeld mit dem Inhalt eines speziellen Speicherbereiches (aus $0E00 bis $0FFF = dezimal 3584 bis 4095) gefüllt, der die Informationen des gewählten Sprites enthält. Falls noch kein Sprite definiert war, kann dabei auch allerlei Byte-Müll abgebildet werden, der uns aber vorläufig nicht erschüttern soll. In der linken oberen Ecke des Rasterbereiches meldet sich ein Kreuz: das ist der Spritecursor. Oben rechts neben dem Raster ist unser Sprite so abgebildet, wie es im Normalbetrieb später auf dem Bildschirm erscheinen wird.

Bevor wir auf die einzelnen Optionen des Sprite-Editors eingehen, noch eine Bemerkung. Alles bisher Beschriebene benutzt den Grafik-Bildschirm. Ebenso wie bei der Verwendung der hochauflösenden und der Mehrfarbengrafik wird dazu der Basic-Anfang nach $4000 verschoben, also über die Bit-Map.

Sehen wir uns nun die Möglichkeiten unseres Editors an:

CLR/HOME: Damit kann das Definitionsfeld gelöscht werden. Das befreit uns vom Byte-Müll.

HOME: Der Sprite-Cursor marschiert in die linke obere Ecke.

CURSOR-TASTEN

Damit wandert der Sprite-Cursor über das Rasterfeld.

RETURN: Der Cursor wird auf den Anfang der nächsten Zeile gesetzt.

A: Die Wiederholungsfunktion der Tasten 1 bis 4 kann durch einmaliges Drücken aus-, durch nochmaliges Drücken wieder eingeschaltet werden.

M: Einmal Drücken schaltet den Mehrfarbenmodus ein. Dadurch bekommen die Tasten 1 bis 4 eine neue Bedeutung.

Nochmaliges Drücken schaltet wieder auf den normalen Hochauflösungsmodus.

X: Ein Druck verdoppelt das Sprite in waagerechter Richtung, ein weiterer Tastendruck erzeugt wieder ein Sprite in normaler Breite.

Y: Mit dieser Taste passiert in der Senkrechten dasselbe wie in der Horizontalen bei der X-Taste.

ZIFFERNTASTEN 1 BIS 4

a) Im Normalmodus setzt
1: die aktuelle Hintergrundfarbe (0),
2-4: die aktuelle Vordergrundfarbe (1).
b)Im Mehrfarbenmodus setzt
1: die Hintergrundfarbe (00)
2: eine Multicolorfarbe (01)
3: die aktuelle Vordergrundfarbe (10)
4: die andere Multicolorfarbe (11)

Zur Erläuterung:

– Im Normalmodus tritt jedes gesetzte Bit (= 1) als Bildpunkt in der Vordergrundfarbe auf, alle nicht gesetzten (= 0) erscheinen in der Hintergrundfarbe.

– Im Mehrfarbenmodus tragen die Bits paarweise zur Farbgebung bei. Deshalb ist hier der Spritecursor auch doppelt so breit.

CONTROL 1-8

Wählt die aktuelle Vordergrundfarben 1-8 aus.

COMMODORE 1-8

Leistet dasselbe mit den Farbcodes 9 bis 16.

Zwei Optionen werden merkwürdigerweise im Handbuch nicht erwähnt, die recht nützlich sind:

C: Das ist eine Kopierfunktion. Nach Tastendruck erscheint die Frage »Copy from ?«. Nach Eingabe einer Spritenummer wird das dazugehörige Muster in das aktuelle Sprite kopiert und dieses dabei überschrieben. Das ist ganz sinnvoll, wenn man mehrere ähnliche Sprites erzeugen möchte.

Was nicht im Handbuch steht

Dasselbe passiert auch, wenn man die Funktionstaste F1 betätigt.

Ist man einmal versehentlich auf diese Tasten geraten, dann kann man durch RETURN wieder in den normalen Editorzustand gelangen.

CONTROL C: Erlaubt das Umschalten zwischen den Sprites. Nach Betätigen dieser Tastenkombination verschwindet die aktuelle Spritenummer. Gibt man nun die gewünschte Nummer ein, schaltet sich das dazugehörige Sprite an.

Zwei Möglichkeiten zum Verlassen des Sprite-Editors sind vorgesehen:

STOP-Taste: Nach dem Betätigen der Taste verliert man alle neuen Eingaben im Definitionsfeld.

Gibt man anschließend eine Spritenummer ein, wird das dazugehörige Muster eingeladen.

Ein anschließendes RETURN führt zum Verlassen des Editors. Mit READY meldet sich der Textbildschirm zurück.

SHIFT-RETURN

Speichern des aktuellen Sprites im Bereich $0E00 - 0FFF.

Ebenso wie bei der STOP-Taste kann man danach durch Eingabe einer neuen Spritenummer das nächste Sprite zur Bearbeitung in den Raster holen.

Ein RETURN führt zum Verlassen des Sprite-Editors.

Interessant an SPRDEF ist, daß man diesen Befehl auch im Programm-Modus verwenden kann. Auf diese Weise kann man im Programm ein oder mehrere Sprites erstellen und anschließend - also nach Aussteigen mit STOP/RETURN oder SHIFT-RETURN/RETURN - läuft das Programm weiter.

Das also sind alle Möglichkeiten unseres Sprite-Editors: Leider fehlen einige nötige Optionen. So ist es nicht möglich, die frisch zusammengebauten Sprites in DATA-Zeilen abzulegen, die Multicolorregister müssen vor dem SPRDEF-Aufruf belegt werden, die Muster kann man nicht invertieren, drehen oder spiegeln.

SPRCOLOR

Im Normalmodus kann die Punktfarbe eines Sprites und auch die Farbe der Bitkombination 10 des Multicolormodus durch den SPRITE-Befehl bestimmt werden oder in SPRDEF durch die Kombinationen von Control- oder Commodoretaste mit einer Ziffer. Die Belegung der Multicolorregister mit Farbe - das entspricht den SPRDEF-Zifferntasten 2 und 4 -, also der Bitkombinationen 01 und 11 erfolgt durch den SPRCOLOR-Befehl. Die Verwendung von

SPRCOLOR A,B

führt in Multicolorregister 1 (Bit-Kombination 01 oder SPRDEF-Taste 2) zur Farbe A, in Multicolorregister 2 (Bit-Kombination 11 oder SPRDEF-Taste 4) zur Farbe B.

Somit ergibt der Befehl

SPRCOLOR 3,6

die Farben ROT und GRÜN, entsprechend den Bitkombinationen 01 und 11.

SPRITE

Dies ist zweifellos der wichtigste Befehl für ein schon vorhandenes Spritemuster. Er schaltet ein Sprite ein und legt seine Eigenschaften fest. Syntax:

SPRITE A,B,C,D,E,F,G

Keine Angst vor den vielen Parametern: Sie müssen nicht immer alle angeben. Zur Bedeutung der einzelnen Buchstaben:

A: Spritenummer (1 - 8)
B = 0: Ausschalten des Sprites
B = 1: Einschalten des Sprites
C: Farbcode für die Vordergrund-

farbe (das ist das Bitpaar 10 bei Multicolorsprites) (1 bis 16)
D: Priorität gegenüber Text:
0 = Sprite vor Text
1 = Text vor Sprite

Nebenbei: Die Priorität von Sprites untereinander wird durch die Sprite-Nummer geregelt: 1 vor 2 vor 3 ... vor 8.

E: X-Vergrößerung:
0 = normale Größe
1 = doppelte Größe
F: Y-Vergrößerung:
0 = normale Größe
1 = doppelte Größe

Sind sowohl E als auch F auf 1 gesetzt, dann erhält man ein vierfach ausgedehntes Sprite.

G: Modus.
0 = normales Hires-Sprite
1 = Multicolor-Sprite

Ein Beispiel soll die Wirkung des Befehls demonstrieren:

SPRITE 1,1,3,0,1,1,0

Dadurch wird Sprite 1 eingeschaltet. Es ist rot, erscheint vor Bildschirmzeichen und ist sowohl in horizontaler wie auch in vertikaler Richtung verdoppelt. Es handelt sich um ein normales (also Hires-Sprite). Nun soll das gleiche Sprite auf die normale Größe gebracht werden:

SPRITE 1,,,,0,0

Es genügt, die Werte anzugeben, die zu ändern sind. Alle anderen werden so übernommen, wie sie vorher festgelegt wurden. Die Spritennummer und die Kommata vor dem zu ändernden Wert sind obligatorisch.

MOVSPR

Das ist ein recht vielgestaltiger Geselle. MOVSPR kommt von "move sprite", also hat dieser Befehl mit der Positionierung und Bewegung von MOBs zu tun. Drei Varianten sind möglich:

a: MOVSPR n,X,Y

Das Sprite mit der Nummer n wird an den Ort mit den Koordinaten X,Y gesetzt. Als Bezugspunkt beim Sprite dient die linke obere Ecke (auch wenn diese unsichtbar ist). Bild 1 zeigt Ihnen das normale Sprite-Koordinatensystem, das in allen Grafik-Modi gültig ist.

Der sichtbare Bildschirm erstreckt sich von X = 24 bis 344 und in der Vertikalen reicht er von 50 bis 250. Für X und Y dürfen Werte zwischen 0 und 65535 eingesetzt werden. Bild 1 skizziert auch, bei welchen Koordinatenangaben ein Sprite voll sichtbar ist:

Links oben müssen dazu die Koordinaten (24/50) gewählt werden. Ein normales Sprite ist bei X = 320, ein in X-Richtung gedehntes bei X = 296 gerade noch voll sichtbar. Entsprechend gilt für die Y-Werte:
Normales Sprite bis Y = 229,
in Y-Richtung gedehntes bis Y = 208.

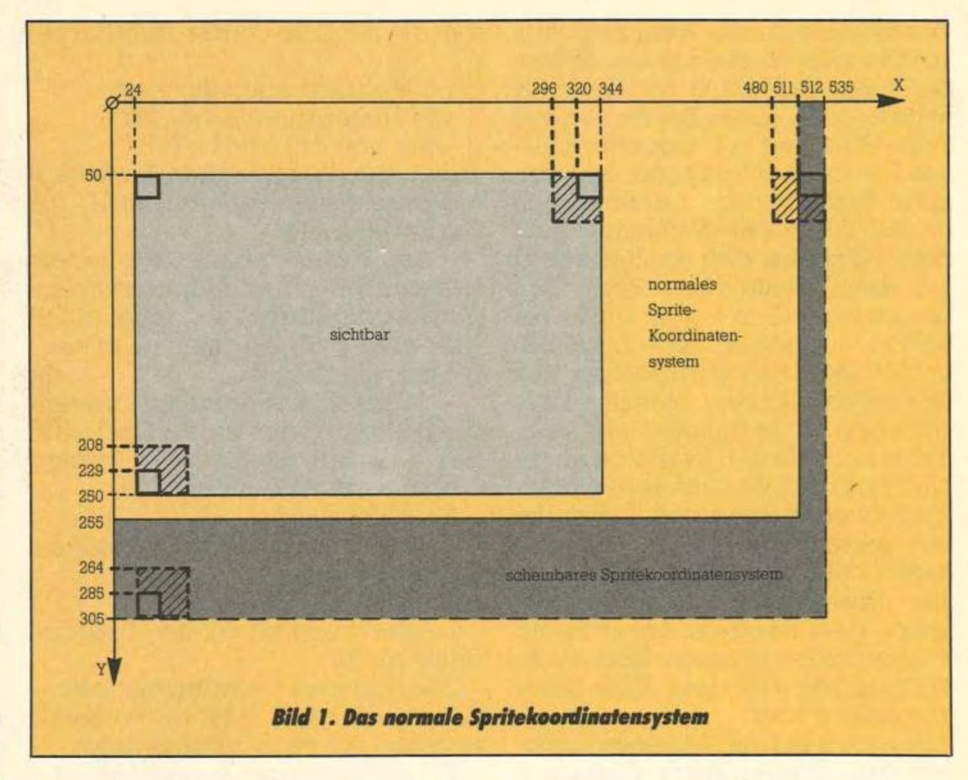


Bild 1. Das normale Spritekoordinatensystem

Größere Koordinatenwerte führen zum Herauswandern aus dem sichtbaren Bereich. Überschreiten die X- oder Y-Werte eine bestimmte Grenze, dann treten die MOBs auf der jeweils gegenüberliegenden Seite wieder ins Bild. Das ist ab X = 513 (bei gedehnten Sprites ab 481) und ab Y = 286 (bei gedehnten Sprites ab 265) der Fall. Eine weitere Erhöhung der Koordinaten führt dann zum Wandern über den Bildschirm, zum erneuten Verschwinden und schließlich Wiederauftauchen auf der anderen Seite und so weiter.

Vorhin war die Rede vom »normalen« Sprite-Koordinatensystem. Daraus kann messerscharf geschlossen werden, daß es auch noch ein anderes gibt, ein »nicht normales«. Das ist tatsächlich der Fall und es handelt sich nicht nur um eines, sondern um eine ganze Menge verschiedener Systeme. Haben Sie nämlich durch den SCALE-Befehl ein neues Hires- oder Multicolor-Bildschirmsystem definiert, dann beziehen sich X und Y auf dieses, was manchmal zu einiger Verwirrung führen kann. Sollte also die Gefahr bestehen, daß ungewollt aus einer früheren Programmphase ein anderes Bildschirmsystem gültig ist, dann kann durch SCALE 0 der normale Zustand hergestellt werden.

Hier noch ein Beispiel:

MOVSPR 1,300,100

setzt Sprite 1 an die Stelle (300/100) im jeweils gültigen Koordinatensystem.

b: MOVSPR n, + /-X, + /-Y

Damit kann man Sprite n relativ um + oder -X (und/oder Y) zur aktuellen Position verschieben. X und Y dürfen wieder Werte bis 65535 annehmen, wobei durch das Vorzeichen nun sogar eine Skalenbreite von -65535 bis +65535 möglich wird. Auch hier beziehen sich X und Y auf das jeweils durch SCALE definierte Koordinatensystem (ohne SCALE-Anwendung also auf das normale Sprite-System aus Bild 1).

Auf diese Weise kann – zum Beispiel in einer Schleife – das Sprite praktisch endlos über den Bildschirm wandern. Es tritt nämlich keine Fehlermeldung auf, wenn die Summe aus alter Position und Verschiebung größer als 65535 wird. Beispielsweise ist es ohne weiteres möglich (ob es sinnvoll ist, wäre eine andere Frage), folgende Kombination zu verwenden:

MOVSPR 8,100,65000:MOVSPR 8,100,+65000

Auch ein Unterlauf unter Null ist auf diese Weise erlaubt.

Wie Sie aus dem Beispiel erkennen können, kann die MOVSPR-Anweisung auch kombiniert verwendet werden:

MOVSPR 1,+10,100 verschiebt Sprite 1 um +10 Einheiten relativ zum vorangegangenen X-Wert und auf die Y-Koordinate 100.

MOVSPR 2,50,-20 läßt Sprite 2 auf die X-Koordinate 50 wandern und



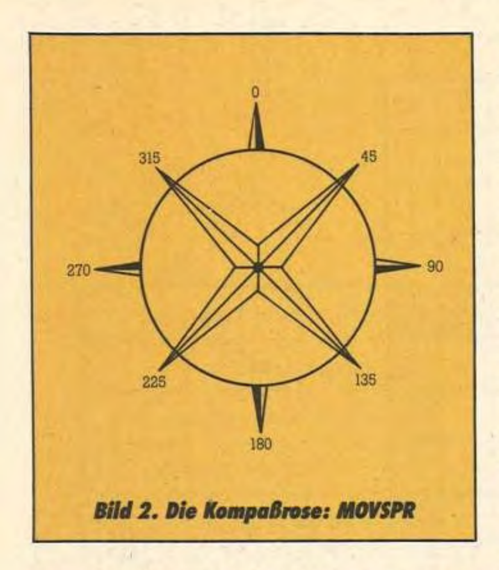


Bild 2. Die Kompaßrose: MOVSPR

gleichzeitig um 20 Y-Einheiten nach oben.

c: MOVSPR n,W # V

Diese Variante des Spritebewegungs-Befehls ist eine feine Sache: Unabhängig vom sonstigen Programmgeschehen wird das Sprite mit der Nummer n mit einem Richtungswinkel W und der Geschwindigkeit V über den Bildschirm gesteuert. Der Winkel W wird in Grad angegeben. Bild 2 zeigt die möglichen Richtungen, die der Kompaßrose entsprechen:

Ohne Fehlermeldung sind Eingaben möglich zwischen -65535 und +65535. Sinnvoll scheint das aber allenfalls in einer Schleife, die ständige Richtungswechsel durchführt, zumal negative Werte und Angaben größer als 33024 keine vernünftige Reaktion ergeben.

Auch für die Geschwindigkeit V sind Angaben zwischen -65535 und 65535 möglich. Als sinnvoll erweisen sich hier nur Werte zwischen 0 und 15. 0 stoppt die Bewegung und 15 ist die höchste erreichbare Geschwindigkeit. Für höhere V-Werte ergibt sich die Geschwindigkeit dann zu modulo(16). (Damit ist der Rest gemeint, der bei Division der eingegebenen Zahl durch 16 verbleibt. So ergibt V=30 dieselbe Geschwindigkeit wie V=14)

Eine Eigenart dieser MOVSPR-Variante ist es, daß die Wirkung dieses Befehls anhält, bis die Bewegung ausdrücklich durch MOVSPR n,W # 0 auf Null gesetzt wird. Auch ein abgeschaltetes Sprite wandert weiter, was Sie leicht mal ausprobieren können.

Der Befehl

MOVSPR 1,90 # 15

läßt Sprite 1 mit Höchstgeschwindigkeit horizontal nach rechts über den Bildschirm jagen.

SPRSAV

Das ist ein sehr interessanter Befehl, dessen Anwendung wir später noch detailliert untersuchen werden. Zwei mögliche Varianten sind vorgesehen:

a: SPRSAV n,A$

Einem String (hier also A$) wird das Bitmuster des Sprite n zugeordnet.

Mittels SPRSAV 1,A$(2)

ordnet man das Bitmuster des Sprite mit der Nummer 1 dem Stringarrayelement A$(2) zu.

b. SPRSAV A$,n

Das ist der umgekehrte Weg: Das in A$ enthaltene Bitmuster definiert nun das Sprite mit der Nummer n.

Durch SPRSAV A$(2),8 wird das in A$(2) gespeicherte Muster in das Sprite Nummer 8 gelesen.

Anstelle von A$ kann jede Stringvariable - auch ein Array-Element - verwendet werden. Einige Möglichkeiten, die auf diese Weise gegeben sind: "Klonen" eines Sprites, Definieren von mehr als acht Sprites (jeweils acht sind gleichzeitig auf dem Bildschirm aktivierbar), schneller Austausch von Spritemustern.

COLLISION

Was im C64-Modus dem Assembler-Programmierer vorbehalten bleibt oder nur durch aufwendige POKE- und PEEK-Operationen realisiert werden kann, ist im C128-Modus mit diesem und dem folgenden Befehl möglich:

COLLISION A,NNNN

fängt drei Typen von Ereignissen ab und setzt die Programmbearbeitung in Zeile NNNN fort. A ist dabei eine Typkennung:

1: Sprite/Sprite-Kollision

2: Sprite/Text-Kollision

3: Lichtgriffelaktivierung

Das Serviceprogramm ab NNNN ist wie ein Basic-Unterprogramm zu behandeln, also auch durch RETURN abzuschließen. Die weitere Bearbeitung geschieht dann an der Stelle, wo bei dem auslösenden Ereignis unterbrochen worden ist.

COLLISION stellt lediglich fest, welcher Typ stattgefunden hat. Bei Spritezusammenstößen kann also damit nicht die Spritenummer identifiziert werden. Es ist erlaubt, mehrere Kollisionstypen gleichzeitig zu aktivieren. Eine Aktivierung innerhalb eines Unterbrechungs-Service-Unterprogrammes ist nicht möglich: Es findet nur die Bearbeitung eines Typs zur Zeit statt. Falls also mehrere aktiviert sind, sollte einer der ersten Befehle im Unterprogramm das Abschalten aller Kollisionsabfragen sein mittels COLLISION A (ohne Zeilennummer). Am Ende der Routine kann dann COLLISION A,NNNN wieder eingeschaltet werden.

Nicht immer ist es sinnvoll, so zu verfahren (also die Kollisionsabfrage am Ende der Routine wieder einzuschalten): Falls beispielsweise der Typ 1 (Sprite/Sprite-Zusammenstoß) als Auslöser definiert ist, muß bedacht werden, daß die Überlappung zweier Sprites einige Zeit andauert und deshalb das Unterprogramm mehrfach angesteuert wird. Im einem Testprogramm (Listing 2) ist dieser Effekt deutlich an den vielen Textwiederholungen zu erkennen.

Unter welchen Umständen wird eine Kollision erkannt? Immer dann, wenn sich sichtbare Teile von Sprites gegenseitig oder mit Bildschirmobjekten (bei Typ 2) überlagern. Abgeschaltete Sprites werden nicht berücksichtigt, wohl aber Zusammenstöße außerhalb des sichtbaren Bereiches (das Handbuch behauptet das Gegenteil).

Probieren Sie doch einmal, in Zeile 40 von Listing 1 die Spritepositionierung:

40 MOVSPR 7,345,100:MOVSPR 8,345,200

Nach dem Start ist keines der beiden Sprites mehr auf dem Bildschirm zu sehen, Kollisionen werden aber gemeldet.

Zwei Aspekte verdienen noch Erwähnung: Zum einen darf man ein Programm, in dem der COLLISION-Befehl verwendet wird, nicht durch STOP unterbrechen. Tut man es trotzdem, dann funktioniert nach einem CONT die Kollisionsabfrage nicht mehr. Zum anderen: Natürlich können alle Sprite-Befehle auch ohne 40-Zeichen-Bildschirm betrieben werden, man sieht nur nichts. Das kann bei COLLISION ganz rätselhafte Effekte erzeugen. So wären Spiele denkbar, die ein Sprite per Joystick durch Hindernisse hindurchbewegen. Das ganze geschieht im Dunkeln (also mit abgeschaltetem oder nicht vorhandenem 40-Zeichen-Bildschirm) und lediglich die Reaktion auf eine Kollision erfolgt auf dem 80-Zeichen-Bildschirm. Der Phantasie sind keine Grenzen gesetzt.

BUMP

Der Name sagt's bereits: Auch hier geht es um Zusammenstöße. Mit dem BUMP-Befehl können wir einfach feststellen, welche Sprites in eine Kollision verwickelt sind:

BUMP(A)

erschließt über die Typkennung A zwei Sorten von Zusammenstößen.

A = 1: Sprite/Sprite-Kollision
A = 2: Sprite/Text-Kollision.

Der Befehl hat lediglich die Aufgabe, den Inhalt eines speziellen Kollisionsregisters auszulesen. Das allerdings macht er so gründlich, daß dieses Register hinterher gelöscht ist. Es empfiehlt sich daher, diesen Wert sogleich in eine Variable zu speichern:

V = BUMP(1)

Die Zahl, die auf diese Weise erhalten wird, muß allerdings erst entschlüsselt werden (siehe da, ein alter Bekannter aus dem C64-Modus), denn jedem Sprite ist ein Bit zugeordnet: Sprite 1 hängt mit Bit 0, Sprite 2 mit Bit 1 zusammen und so weiter bis zu Sprite 8, welches mit Bit 7 geht. Findet nun die Kollision statt, dann schalten die Bits der kollidierten Sprites auf »1«. Bild 3 zeigt Ihnen diese Zusammenhänge und einige Rechenbeispiele.

Bits:	7	6	5	4	3	2	1	0	
Bitwerte:	128	64	32	16	8	4	2	1	
Sprites:	8	7	6	5	4	3	2	1	
Beispiele:	0	0	0	0	0	0	1	1	= 3, also Sprites 1 und 2
	0	0	0	0	0	1	0	1	= 5, also Sprites 1 und 3
	0	0	0	0	1	0	0	1	= 9, also Sprites 1 und 4
	1	0	0	0	0	0	0	1	= 129, also Sprites 1 und 8
	1	1	0	0	0	0	0	0	= 192, also Sprites 7 und 8

Bild 3. Das Ergebnis der BUMP(A)-Abfrage

Ein Supercrash unter Beteiligung aller acht Sprites würde dann also die Zahl 255 ergeben. Ein kleines Testprogramm (Listing 3) soll das Vorgehen dabei illustrieren.

Besonders die Zeilen 90 und 100 sind sicherlich für Sie interessant, weil man mit ihnen die Nummern der beteiligten Sprites feststellen kann.

Auch hier gilt, was schon bei COLLISION bemerkt wurde: Auch Zusammenstöße außerhalb des sichtbaren Bildschirmes werden registriert und das auch ohne 40-Zeichen-Bildschirm. Ebenso wie dort muß beachtet werden, daß ein Zusammenstoß einige Zeit in Anspruch nimmt, also die Reaktion darauf meistens mehrfach erfolgt.

Im Gegensatz allerdings zu COLLISION, das interrupt-gesteuert abläuft, muß die Abfrage des BUMP-Wertes mehrfach erfolgen. Der Befehl ist sowohl ohne als auch mit COLLISION einsetzbar. Im letzteren Fall dient er als Ergänzung im Unterprogramm zur Feststellung der Spritenummern.

RSPCOLOR

Damit sind Informationen zu den aktuellen Multicolor-Register-Inhalten zu bekommen:

RSPCOLOR(A)

liefert bei

A = 1: Farbcode in Multicolorregister 1 (das ist das, welches der Bitkombination 01 zugeordnet ist)

A = 2: Hier wird der Inhalt des anderen Multicolorregisters ausgegeben, das zu der Bitkombination 11 gehört.

RSPPOS

Sollte uns die aktuelle Position oder Geschwindigkeit eines Sprites interessieren, dann können wir das durch

RSPPOS(n,A)

erfahren. Dabei ist n die Spritenummer und A wieder eine Kennung mit folgender Zuordnung:

A = 0: aktuelle X-Koordinate
A = 1: Y-Koordinate
A = 2: gerade vorhandene Geschwindigkeit

Leider gibt es da eine kleine Unstimmigkeit gegenüber dem Befehl MOVSPR: Gleichgültig, welches Sprite-Koordinatensystem wir dort durch SCALE festgelegt haben, RSPPOS liefert immer die Werte des normalen Systems. RSPPOS(0) liegt daher immer zwischen 0 und 511, RSPPOS(1) immer zwischen 0 und 255. Sollten Sie bei normalem System mittels MOVSPR höhere Werte eingegeben haben, dann wird hier immer modulo(512) für die X- und modulo(256) für die Y-Koordinate ausgegeben.

Auch die Sache mit der aktuellen Geschwindigkeit hat einen Haken: Falls Sie nämlich den Sprite mal auf andere Weise als durch MOVSPR n,W # V bewegen, erhalten Sie den Wert 0 als Ausgabe von RSPPOS(2).

RSPRITE

Hier haben wir das Pendant zum SPRITE-Befehl vorliegen. Alle dort verwendeten Parameter können hier durch

RSPRITE(n,A)

abgefragt werden. n ist wieder die Spritenummer, A eine Kennung mit den Zuordnungen:

A = 0: Sprite ein- (= 0) oder ausgeschaltet (= 1)
A = 1: Farbcode des Sprites (Im Multicolormodus Farbe der Bitkombination 10)
A = 2: Sprite vor (= 0) oder hinter (= 1) Text.
A = 3: In X-Richtung gedehnt (= 1) oder nicht (= 0).
A = 4: In Y-Richtung gedehnt (= 1) oder nicht (= 0).
A = 5: Multicolormodus (= 1) oder normales Hires-Sprite (= 0)

JOY

Dieser Befehl hat zwar nicht unbedingt etwas mit den Sprites zu tun, wird aber häufig zum Steuern der Sprites eingesetzt und kommt daher an diese Stelle. Man kann damit auf einfache Weise den Zustand der Joystickports abfragen:

JOY(A)

fragt bei A = 1 den Port 1, und bei A = 2 den Port 2 ab.

Bevor wir uns die Bedeutung der Abfrageergebnisse ansehen, noch eine Warnung: Offensichtlich entspricht der Feuerknopf in Port 1 der Taste F8, so daß im Direktmodus immer der Maschinensprachemonitor angesprungen wird, wenn man den Feuerknopf drückt. Beim Port 2 gibt es dieses Problem nicht.

Die mittels JOY ausgelesenen Werte haben folgende Bedeutung:

0 = Ruhestellung
1–8 = Richtungen (siehe Bild 4)
128 = Ruhestellung, Feuerknopf gedrückt
129–136 = Richtungen mit gedrücktem Feuerknopf (siehe Bild 4)

Auf diese Weise können wir bequem Sprites mit dem Joystick steuern:

100 ON JOY(2) GOSUB 300,310,320, 330,...

Im Testprogramm (Listing 4) ist eine mögliche Variante dazu gezeigt.



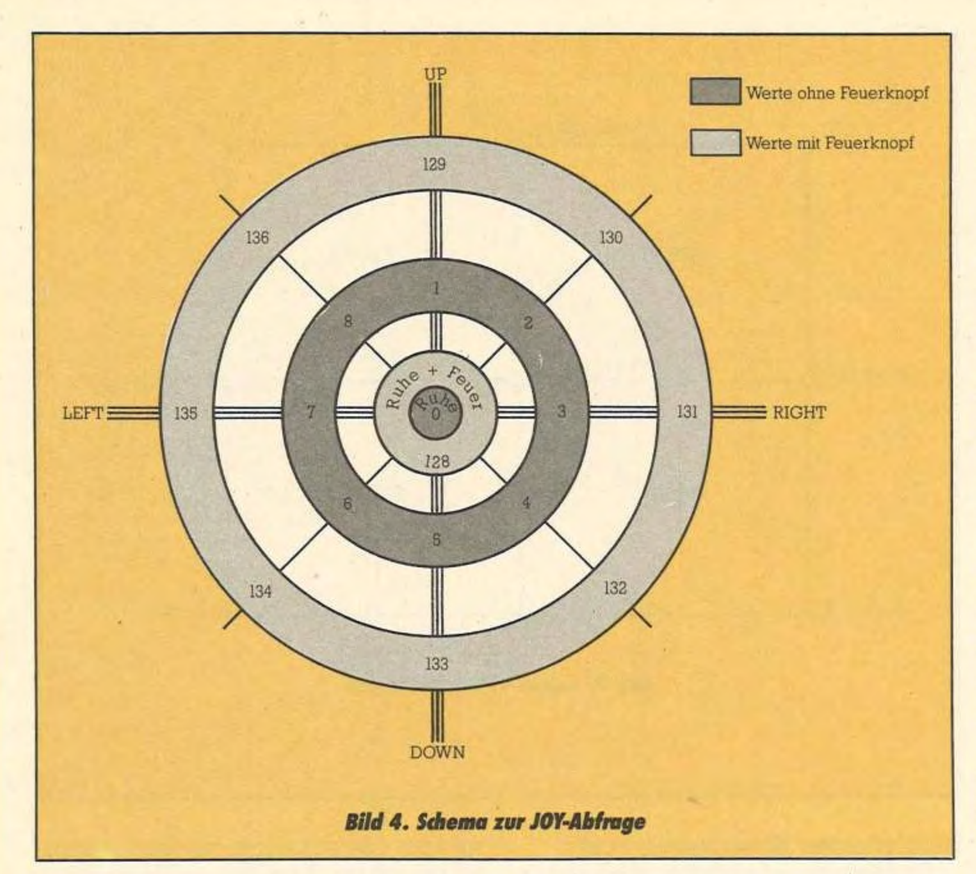


Bild 4. Schema zur JOY-Abfrage

Damit haben wir alle im Basic 7.0 enthaltenen Sprite-Befehle behandelt. Allerdings sind noch einige Fragen offen geblieben, die uns nun weiter beschäftigen werden.

Wie kann man Sprites der Nachwelt erhalten?

1. Sprites im Speicher

a. Sprite-Daten-Speicher

Schon beim SPRDEF-Befehl haben wir diesen Speicherbereich zwischen 3584 und 4095 ($0E00 bis 0FFF) erwähnt, in dem die Spritedaten landen, wenn ein Sprite durch Shift-Return gespeichert wird. Pro MOB befinden sich dort - in der Reihenfolge der Spritenummern - je 63 Byte und eine Endmarkierung (das ist ein 0-Byte). Verfügbar bleiben diese Daten, bis sie überschrieben oder der Computer abgeschaltet wird.

b. Sprite-Strings

Noch flüchtiger ist die Existenz der Daten im String, der durch SPRSAV gebildet wird. Schon ein CLR macht der Kunst den Garaus. In jeweils 67 Byte (die letzten vier enthalten immer die Werte 23, 0, 20, 0) finden sich dort die Bitmuster. Sollten Sie einmal versuchen, so einen String (z.B. A$) durch PRINT A$ auf dem Bildschirm zu zeigen, dann müssen Sie sich auf allerhand Überraschungen gefaßt machen: Alle Bytewerte von 0 bis 255 können auftreten und wirken so, als wären sie mittels CHR$(Wert) zur Ausgabe aufgerufen worden. Wenn also der Wert 147 zufällig dabei sein sollte, dann wird der Bildschirm gelöscht, etc. Jedes RUN führt übrigens ebenfalls ein CLR aus. Auch das ist daher nicht die richtige Methode, die mühselig konstruierten Sprites etwas länger am Leben zu erhalten.

Sowohl im Speicher als auch im String haben die Sprite-Daten genau dasselbe Format (bis auf die letzten Bytes, die im einen Fall aus einer 0, im anderen aus den Zahlen 23,0,20,0 bestehen). Das können Sie selbst nachprüfen am Programm "VERGLEICH SPRDT" (Listing 5).

Nach dem Start werden zunächst die Spritedaten von der Diskette geladen und dann in Strings eingelesen. Anschließend erscheinen auf dem Bildschirm Sprite für Sprite die Bytewerte sowohl aus dem Speicher als auch aus dem String.

2. Sprites auf Diskette und Kassette.

a. BSAVE

Eine elegante Methode zur Verewigung unserer Sprites ist das Speichern des Spritespeichers mittels des BSAVE-Befehls. Man schreibt dazu:

BSAVE"Name",ON B0,P3584 TO P4095

Damit sind dann alle acht Sprites erfaßt. Leider arbeitet dieser Befehl nicht mit der Kassettenstation, denn es sind nur Gerätenummern von 4 bis 15 zulässig. Das Wiedereinladen geschieht dann (wie in den hier vorgestellten Beispielprogrammen) mittels:

BLOAD"Name",ON B0

b. Vom Monitor aus

Damit ist es nun auch Benutzern der Datasette möglich, den Spritespeicher auf Kassette zu sichern. Der Monitor verfügt über ein Kommando S zum Speichern beliebiger Speicherbereiche. Mittels MONITOR oder der Funktionstaste F8 schalten Sie den Monitor an, dann verwenden Sie:

S"Name",01,+3584,+4095 (Kassette) oder

S"Name",08,+3584,+4995 (Diskette).

Mit Hilfe des L-Kommandos im Monitor kann solch ein File dann problemlos wieder geladen werden:

L"Name",01(oder 08),+3584

c. Als sequentielles File

Hat man die Sprites in Strings abgelegt (das dürfen dann auch mehr als acht sein), dann kann man sich der üblichen Techniken zur Speicherung in sequentiellen Dateien bedienen. Allerdings kann es manchmal dabei Schwierigkeiten mit bestimmten Byte-Inhalten geben. So ist es durchaus möglich, daß ein Byte zufällig den Inhalt 13 (also einem RETURN entsprechend) hat, was zu Störungen beim Wiedereinlesen führen kann. In solchen Fällen könnte man die einzelnen Bytes (wie im Programm VERGLEICH SPRDT geschehen) in die ASCII-Zahlen wandeln und in dieser Form als SEQ-File speichern. Diese Möglichkeit werden wir nicht weiter beschreiben, weil die Speicherung per Monitor oder BSAVE schneller und effektiver erscheint.

3. Für ein Listing

Viele Programme werden nicht in Form von Disketten- oder Kassettenfiles, sondern einfach auf dem Papier weitergegeben. Dieses Problem lösen bessere Sprite-Editoren durch eine Funktion, die alle Werte in DATA-Zeilen an ein Programmende anhängt. In SPRDEF existiert diese Möglichkeit leider nicht, was uns dazu zwingt, selbst die Initiative zu übernehmen.

a. Aus dem Speicher in DATAs

Dazu haben wir ein kleines Programm »SPRITEDATAS« gestrickt (Listing 6), das Sie mit der MERGE-Funktion an Ihr eigenes fertiges Spriteprogramm anhängen können. Noch eine Warnung: Wenn Sie SPRITDATAS abgetippt haben, speichern Sie es unbedingt vor

einem RUN ab, denn es verabschiedet sich am Ende des Programmlaufes aus Ihrem Basicspeicher. Nun also zur Verwendung des Programmes. Man startet es durch RUN63000. Das Auslesen des Sprite-Datenspeichers geschieht im FAST-Modus (der 40-Zeichen-Bildschirm verabschiedet sich vorübergehend), danach erscheinen jeweils zwei Programmzeilen (mit Zeilennummern ab 63020) und darunter ein GOTO 63009. Außerdem meldet sich ein BREAK und READY. Mittels der HOME-Taste und 3 aufeinanderfolgenden RETURNs übernehmen Sie die Zeilen ins Programm und erzeugen den nächsten Bildschirm, wo das Spielchen dann ebenso weitergeht. Insgesamt machen wir das achtmal (für 8 Sprites). Zu guter Letzt wird in Zeile 63040 noch eine Einleseschleife übernommen und es meldet sich der Befehl DELETE 63000-63011. Haben Sie auch das mit HOME und zwei RETURNs übernommen, dann ist der DATA-Generator gelöscht und an Ihr Programm wurde die gesamte DATA-Sequenz angehängt. Als letzte Zeile finden Sie beim Listen noch die Einleseschleife, zu der Sie nun nur noch an passender Stelle Ihres Programmes mittels GOSUB 63040 springen müssen.

Es gibt sicherlich elegantere Möglichkeiten, DATA-Zeilen zu generieren. So könnten wir auch den Tastaturpuffer verwenden. Das aber überlassen wir Ihrer Schöpferkraft.

b. Aus den Strings in DATAs

Auch das Übertragen der Stringinhalte in DATA-Zeilen ist eine Lösungsmöglichkeit unseres Problems. Sie ist sogar sehr verlockend, weil wir damit auf einen Streich mehr als acht Sprites ins Listing schreiben könnten. Die Programmierung wird aber etwas komplizierter, weil hier nicht mehr der ganze Inhalt des Sprite-Strings in einen Zeilen-String gelesen werden kann. Es gibt eine Reihe von Strategien um das Problem zu lösen (mehr oder weniger einfache, was teilweise mit dem Auslesen der DATAs zusammenhängt), auch hier sind Sie gefordert!

Koppeln von Sprites

Sollten Sie ein komplexeres grafisches Gebilde erzeugen wollen als es mit einem Sprite möglich ist, dann können Sie auch mehrere Sprites koppeln. Am Beispiel von vier Sprites zeigt Ihnen Bild 5 die dazu nötigen Koordinatenwerte:

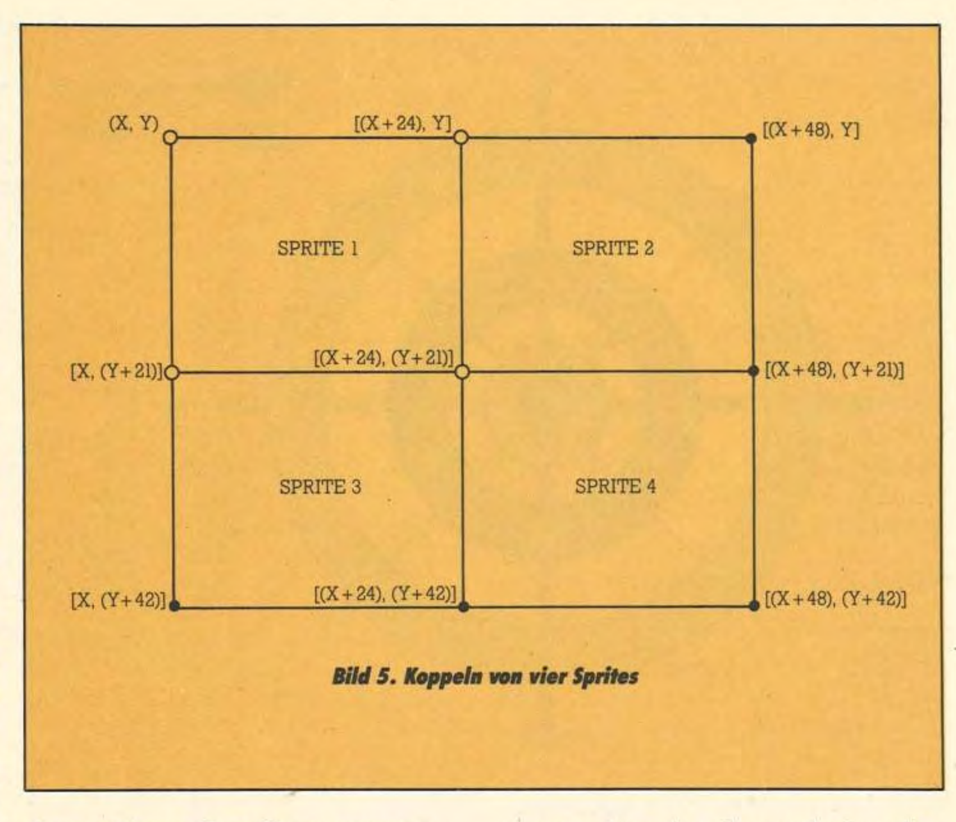


Bild 5. Koppeln von vier Sprites

Das Prinzip ist einfach: Weil jedes Sprite in X-Richtung 24 Bit breit ist, ergibt sich für die X-Koordinate des Nachbarsprites bei nahtloser Verbindung ein um 24 größerer Wert. 21 Bit mißt ein MOB in der Vertikalen, wodurch sich Y+21 für ein unterhalb des ersten gelegenes Sprite berechnet als Y-Koordinate. Das Programm »VIER SPRITES« (Listing 7) demonstriert die Verhältnisse.

Zunächst wächst ein grafisches Objekt aus vier Sprites zusammen. Dieses kann dann mit dem Joystick in Port 2 (nach Druck auf den Feuerknopf) über den Bildschirm gesteuert werden (Zeile 300). Ein weiterer Druck auf den Feuerknopf beendet das Programm. Wenn Sie bei bestimmten Richtungen genau hinsehen, werden Sie eine leichte Unstimmigkeit in der Bewegung feststellen. Bei noch größeren Objekten scheint Basic für diese Steuerung doch etwas zu langsam zu werden und man muß auf Maschinensprache umsteigen um eine flüssige Bewegung zu erreichen.

Mehr als acht Sprites

Hier soll nicht die Rede sein von trickreichen Anwendungen der Unterbrechungsprogrammierung per Assembler, die es tatsächlich zuläßt, mehr als acht Sprites GLEICHZEITIG auf dem Bildschirm zu zeigen. Vielmehr geht es uns um die schnelle, nacheinander ausgeführte Abbildung. Sehen Sie sich dazu zuerst einmal das Programm »SPRITE-TRICK« (Listing 8) an. Hier liegen acht Spritemuster vor, die nacheinander – mit programmierter Verzögerung, weil's sonst zu schnell ginge – auf derselben Bildschirmstelle gezeigt werden. Es ergibt sich ein kleiner Trickfilm.

Nun sind acht Teilbilder ein etwas ärmlicher Bewegungseffekt. Um wirklich längere Passagen zu zeigen, müßten wir anders verfahren. Das soll im Programm »24 SPRITES« demonstriert werden.

Das Geheimnis – Sie haben es sicherlich schon längst vermutet – liegt im SPRSAV-Befehl begründet. 24 Spritemuster wurden hier in ein Stringarray eingelesen (Zeilen 20 bis 80). Während die Sprites auf dem Bildschirm aktiv sind (die Aktivierung erfolgt schon in den Zeilen 130 bis 150), wechseln wir durch die Zuordnung anderer Spritemuster aus den Strings ihre Erscheinungsformen. Das geschieht in der DO...LOOP Schleife ab Zeile 170. Hier geschieht das – wegen des hübschen Mustereffektes – zyklisch (durch Drücken von »—« können Sie das Programm beenden), man kann sich aber ohne weiteres vorstellen, wie ein oder mehrere große Arrays gebildet und dann fortlaufend deren Spritemuster verwendet werden können. Gleichzeitig sind die sich ändernden Sprites noch beweglich...



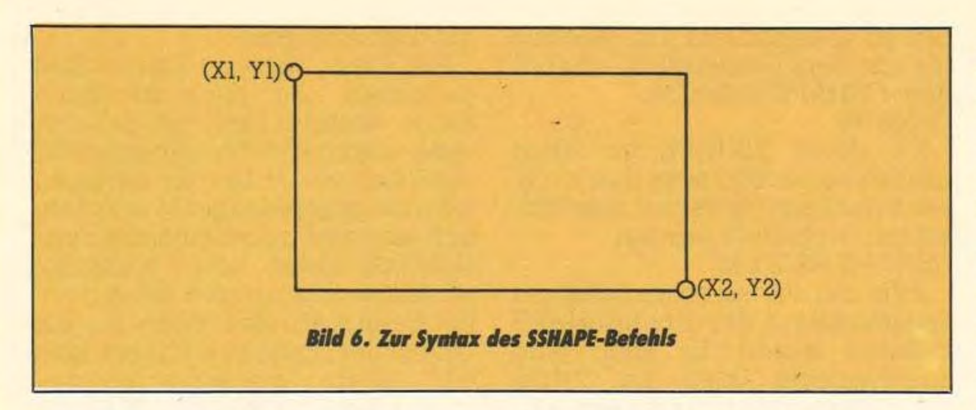


Bild 6. Zur Syntax des SSHAPE-Befehls

Sie sehen, daß kaum Grenzen gesetzt scheinen, allerhand Interessantes auf dem Bildschirm ablaufen zu lassen.

Noch ein paar Sprite-Besonderheiten

1. Sprites und Splitscreens

Beim C128 haben wir ja in den Grafik-Modi 2 und 4 die Möglichkeit, sowohl hochauflösende (oder aber Multicolor-) Grafik als auch Text auf einem Bildschirm gleichzeitig darzustellen. Damit ist nicht etwa der CHAR-Befehl gemeint, sondern durch eine spezielle Technik, die den Rasterzeileninterrupt ausnutzt, wird der Bildschirm tatsächlich in zwei verschiedenen Modi betrieben.

Wie verhalten sich nun die Sprites auf solch einem Bildschirm? Wie es Kobolden ansteht, scheren sie sich überhaupt nicht darum! Probieren Sie mal das Programm VIER SPRITES mit einer kleinen Änderung aus: In die Zeile 130 fügen Sie noch ein GRAPHIC 2,0,12 (oder für den Multicolormodus GRAPHIC 4,0,12) und starten dann. Ohne Reaktion von Seiten der Sprites können Sie diese über die in der Bildmitte liegende Modusgrenze hinwegsteuern. Allerdings reagiert der Bildschirm mit einem penetranten Flattern der Grenzlinie, wenn das Sprite genau darauf liegt.

Falls Sie dasselbe mit dem Grafik-Modus 4 (also GRAPHIC 4,0,12) probieren, würde es uns interessieren, ob auch Ihr Computer nach einiger Zeit mit irgendeiner Fehlermeldung aussteigt. Beim Modell des Autors hat er sich so verhalten. Wenn er sich dann die als fehlerhaft gemeldete Zeile listen ließ, befanden sich allerlei merkwürdige – anscheinend zufällig erzeugte – Fehler darin. Zuvor aber war die Zeile fehlerfrei gewesen! Man sollte offensichtlich diese Möglichkeit mit etwas Vorsicht gebrauchen.

2. Sprites und Windows.

Ebensowenig wie sich Sprites um den Splitscreen kümmern, scheren sie sich um Windows. Probieren Sie es mal aus, indem Sie in Zeile 130 des Programmes »VIER SPRITES« noch ein Window definieren (zum Beispiel mit WINDOW 5,5,20,10). Unbeirrt von allen Einschränkungen, denen unser Text unterworfen ist (hier die Aufforderung zum Benutzen des Port 2), läßt sich unser Spritegebilde kreuz und quer verschieben.

Die Sprites an sich sind damit zunächst einmal durchleuchtet. Legen wir sie also für eine Weile beiseite und wenden wir uns den anderen Grafikobjekten des C128 zu, den Shapes.

Was sind Shapes?

Wie so vieles aus der Computerkultur stammt auch dieses Wort aus dem angelsächsischen: Shape heißt ins Deutsche übersetzt soviel wie Form, Gestalt, Umriß. Hier bezeichnet Shape einen genau definierten Bildschirmausschnitt, der gespeichert und wiederverwendet werden kann. Wie das mit dem C128 geschieht, soll nun erklärt werden.

SSHAPE

Dies ist der Befehl, mit dem ein Bildschirmbereich in einen String eingelesen werden kann. Seine Syntax ist

SSHAPE A$,X1,Y1,X2,Y2

A$ ist hier dann eine beliebige Stringvariable, auch ein Array kann Verwendung finden. X1 bis Y2 bezeichnet die Eckkoordinaten des Rechteckes, dessen Inhalt als Shape definiert werden soll. Bild 6 soll das etwas verdeutlichen:

(X1,Y1) gehören zum linken oberen und (X2,Y2) zum rechten unteren Eckpunkt. Es ist auch möglich, X2 und Y2 wegzulassen. In diesem Fall werden durch den SSHAPE-Befehl einfach die aktuellen Koordinaten des Grafik-Cursors eingesetzt.

Falls es noch unklar sein sollte: Im Gegensatz zu Sprites, die jedem Bildschirm-Modus trotzen (also sowohl als Hires- oder Multicolorsprites im Text-, als auch im Hochauflösungs- und im Multicolormodus auftauchen können), sind Shapes fest mit dem aktuellen Modus verbunden.

Ein Shape ist ein Grafikobjekt, kann daher auch nur in den Grafikmodi auftreten und definiert werden. Also in denen, die durch GRAPHICn erzeugt werden, wobei n von 1 bis 4 geht. Außerdem ist ein Multicolorshape auch nur im Multicolormodus ein solches. Schaltet man beispielsweise mittels GRAPHIC1 um in den normalen Hochauflösungsmodus, wird auch unser Shape nur als hochaufgelöstes sichtbar.

Wie wir uns aus all dem schon fast denken können, ist das Koordinatensystem, auf das sich die Angaben X1 bis Y2 beziehen, das im jeweiligen Modus gültige. Im Normalfall also im Hires-Modus X von 0 bis 319 und Y von 0 bis 199. Im Multicolormodus geht X nur von 0 bis 159.

A$ ist ein String. Für Strings gilt beim C128 die Begrenzung auf maximal 255 Byte. Ein durch SSHAPE in einen String zu schreibendes Gebiet darf also eine gewisse Größe nicht überschreiten. Bezeichnen wir als

DX = X2 – X1

und als

DY = Y2 – Y1

dann gilt für die Bytemenge im String im Hochauflösungsmodus die Formel:

Lh = INT((ABS(DX) + 1)/8 + .99)*(ABS(DY) + 1) + 4

Im Multicolormodus gilt stattdessen die Gleichung:

Lm = INT((ABS(DX) + 1)/4 + .99)*(ABS(DY) + 1) + 4

Die Addition von 4 am Ende der Formeln ergibt sich durch vier Byte, die den Schluß des Shape-String bilden und Steuergrößen enthalten.

Natürlich können Sie jedesmal, wenn Sie ein Shape erstellen, von dem Sie argwöhnen, es könne zu groß sein für den String, unsere Formeln anwenden. Eine Hilfe soll Ihnen Listing 9 »SSHAPE-TABELLE« geben. In der vorliegenden Form druckt es eine Liste aus, in deren linker Spalte Werte für eine Y-Differenz stehen. Rechts daneben sind die maximal zulässigen DX-Werte angegeben sowie die sich ergebende Stringlänge. Auf diese Weise können Sie – ausgehend von DY – mit einem Blick erkennen, wie groß DX sein darf. Es empfiehlt sich, noch drei weitere Tabellen auszudrucken. Da ist zunächst einmal

interessant, wie der Maximalwert von DY bei gegebenem DX aussieht. Zu diesem Zweck ist lediglich die Reihenfolge des Ausdruckes und der Rechenoperationen zu ändern:

In Zeile 20 kommt dann X-DIFF vor Y-DIFF, Zeile 40 enthält X=0, in Zeile 50 schreiben wir DO UNTIL X=320, in der nächsten Zeile 60 steht: X=X+1:Y=0. Zeile 70 wird zu DO UNTIL Y=200 und Zeile 80 zu Y=Y+1. Schließlich wird auch in Zeile 90 noch die Reihenfolge des Ausdruckes umgestellt zu PRINT #1,X1,Y1,Z.

Diese beiden Tabellen beziehen sich auf Shapes im Hochauflösungsmodus. Zwei weitere Tabellen für den Multicolormodus bekommen Sie, wenn Sie in Zeile 30 die vorhin angegebene Berechnungsfunktion zu diesem Modus einfügen. Nur die Zahl 8 muß dazu in eine 4 verändert werden.

Sollte dann, trotz all dieser Berechnungsmöglichkeiten – man sieht ja nicht jedesmal in die Tabellen – der zu speichernde Shape größer werden als 255 Byte, dann meldet unser Computer einen STRING TOO LONG ERROR.

Einen Aspekt zum SSHAPE-Befehl müssen wir noch besprechen, weil es da einige Verwirrung geben kann: Wie reagiert SSHAPE auf ein durch SCALE verändertes Bildschirmsystem? Die Koordinatenangabe bei SSHAPE muß dann ebenfalls in den aktuell gültigen Werten geschehen. Man muß sich also nicht um die Organisation des Koordinatensystems kümmern, denn durch SSHAPE wird automatisch die Umrechnung auf den normalen Bildschirm vorgenommen. Lediglich bei der Berechnung der Stringlänge müssen einige Korrekturen eingefügt werden.

Nehmen wir an, wir hätten durch SCALE 1,XM,YM ein System definiert, dann liegen (im Hochauflösungsmodus) die Vergrößerungsfaktoren

FX = XM/320
FY = YM/200

vor und die Beziehungen für die Stringlänge lauten dann:

Hochauflösungsmodus:
Lh=INT((ABS(DX*320/XM)+1)/8 +.99)* (ABS(DY*200/YM)+1)+4

Multicolormodus:
Lm=INT((ABS(DX*160/XM)+1)/4 +.99)* (ABS(DY*200/YM)+1)+4

DX und DY sind dabei im aktuellen Koordinatensystem geltende Werte.

Begnügen wir uns nun mit dem SSHAPE-Befehl (es gäbe noch einiges zu untersuchen) und wenden wir uns dem Gegenstück, nämlich dem GSHAPE-Befehl zu.

GSHAPE

Ein durch SSHAPE im String gespeichertes Bild kann durch diesen Befehl nun wieder auf dem Bildschirm dargestellt werden:

GSHAPE A$,X,Y,M

A$ ist der uns nun schon bekannte Speicherstring, der durch SSHAPE definiert wurde. Es kann jede Stringvariable oder ein String-Array-Element verwendet werden.

X,Y sind die Koordinaten im aktuellen System, an die die linke obere Ecke unseres Shapes gelegt werden soll.

Interessant ist die Angabe von M. Hier dreht es sich um einen Darstellungsmodus, von dem es hier vier Möglichkeiten gibt:

M = 0: Unser Shape wird genauso abgebildet, wie es definiert wurde. Schon an dieser Stelle auf dem Bildschirm vorhandene Objekte werden überdeckt. Dasselbe ergibt sich, wenn wir diesen Parameter einfach weglassen.

M = 1: Das Shape wird invertiert abgebildet. Auch hier überdeckt das Shape vorhandene Objekte.

M = 2: OR-Modus: Bildpunkte werden sichtbar, wenn sie zum Shape oder zu vorhandenen Objekten gehören. Beide Bilder überlagern sich so.

M = 3: AND-Modus:Bildpunkte werden nur dann gesetzt, wenn sie sowohl zum Shape als auch zum Bildschirmobjekt gehören.

M = 4: EOR-(oder XOR-) Modus: Bildpunkte werden nur dann gesetzt, wenn das entsprechende Bit für Shape und Bildschirmobjekt ungleich ist.

Die Modi 2, 3, 4 sind etwas schwer vorstellbar. Zur Verdeutlichung finden Sie hier ein kleines Demonstrationsprogramm namens »GSHAPE-MODI« (Listing 10). Hier wird zuerst ein Shape definiert, dann der gesamte Bildschirm quergestreift und schließlich das soeben erstellte Shape in allen fünf Modi darauf dargestellt. Die Wirkung ist auf diese Weise ganz gut zu erkennen, besonders bei den Modi 2, 3 und 4.

Der Modus 4 ist aufgrund der EOR-Behandlung interessant: Wird nämlich auf dieselbe Stelle noch einmal das Shape im Modus 4 gezeichnet, dann verschwindet es ganz und der Bildschirmhintergrund ist wieder vorhanden. Will man Shapes bewegen, ohne Bildschirmobjekte zu zerstören, dann ist diese Darstellungsweise ganz gut dazu geeignet.

Die Frage, wieviele Shapes man definieren und auch darstellen könne, erübrigt sich fast: Beliebig viele, solange der Speicherplatz für die Strings reicht. Und davon haben wir eine ganze Menge! Man könnte sich also eine ganze Bibliothek von Hochauflösungs- oder Multicolorobjekten zulegen und diese dann bei Bedarf abrufen. Sollte für ein Objekt der String zu kurz sein, können - ebenso wie wir es bei den Sprites schon praktiziert haben - mehrere Shapes gekoppelt werden. Sie erkennen schon: Auch hier sind die Möglichkeiten sehr breit.

Zusammenspiel von SSHAPE und GSHAPE.

Eine Frage haben wir noch offen gelassen, die eine recht verlockende Konsequenz in sich birgt: Wenn man mittels der SCALE-Anweisung die Systeme bei SSHAPE (also beim Aufbau des Shapes) und bei GSHAPE (bei der Abbildung) unterschiedlich wählt, kann man dann Shapes verkleinern oder vergrößern?

Die Antwort ist leider nein. Bei GSHAPE werden nur die Koordinaten eines (des linken oberen) Eckpunktes angegeben. Lediglich auf den Darstellungsort hat somit ein unterschiedliches Koordinatensystem eine Wirkung, nicht aber auf die Shape-Größe.

Bewegen von Shapes

Im Gegensatz zu den Sprites ist das Bewegen von Shapes eine langweilige Angelegenheit. Jedem Shape-Aufbau durch GSHAPE kann man ganz geruhsam zusehen. Zwar ist es durch Sequenzen wie:

```
FOR I = 0 TO 200
    GSHAPE A$,I,I,4
    GSHAPE A$,I,I,4
NEXT I
```

möglich, Shapes ohne Schaden für den Bildschirminhalt über den Sichtbereich ziehen zu lassen. Das ganze ähnelt aber bei weitem nicht der Sprite-Bewegung. Aus diesem Grund ist es anzuraten, bei der Bewegung von Bildschirmobjekten – wann immer möglich – Sprites zu wählen.

Shapes der Nachwelt erhalten

Wir stehen bei den Shapes demselben Problem gegenüber wie bei den Sprites. Auf welche Weise kann ein einmal erstelltes Shape für weiteren Gebrauch beiseitegelegt werden?

1. Shapes im Speicher

a. Der Shape-String

Durch SSHAPE wurde ein String geschaffen, der die gleiche Lebensdauer hat wie jede andere

Variable: RUN und CLR löschen ihn.

b. Shape im Sprite-Speicher

Wir werden später eine Möglichkeit kennenlernen, die es unter gewissen Umständen erlaubt, ein Shape zum Sprite zu machen. Auf diese Weise gelangt das Muster dazu dann in den Speicherraum, der den Sprites vorbehalten ist und überlebt, bis der Computer abgeschaltet oder der Speicher überschrieben wird.

2. Shapes auf Diskette oder Kassette

Damit wird es nun wirklich interessant. Der Massenspeicher kann Shapes »auf ewig« festhalten.

a. Shape-Strings direkt

Im Grunde genommen sollte es möglich sein, den durch SSHAPE definierten String direkt als sequentiellen File auf Diskette oder Kassette abzulegen und zu lesen. Allerdings ergeben sich hier – wie auch schon bei den Sprite-Strings – Schwierigkeiten beim Lesen. Weil im Prinzip alle Bytewerte von 0 bis 255 auftreten können, also auch ein Wert, der einem RETURN oder einem Anführungszeichen entspricht, wird der String nicht immer vollständig und fehlerfrei gelesen. Dazu kommt das Problem, daß die Shape-Strings unterschiedliche Längen aufweisen können. Wie bestimmt man das Ende eines solchen Files?

b. Als ASCII-Werte speichern

Ebenfalls bei den Sprites wurde diese Möglichkeit schon angedeutet: den Shape-String auseinander zu nehmen und jedes Byte in Form seines ASCII-Wertes zu speichern. Eine Variante dieses Verfahrens zeigt Ihnen das Listing 11 »SHAPE SPEICHERN«. Hier wird zuerst ein Shape (einfach ein Kreis) erzeugt und in A$ abgelegt. Ein Unterprogramm druckt auf dem Textbildschirm (für Benutzer von 2 Bildschirmen sofort, für andere erst später sichtbar) die ASCII-Werte des Strings aus. In Zeile 40 wird der alte File SHAPE1 gelöscht und dann ab Zeile 50 der neue auf Diskette abgelegt. Benutzen Sie eine Datasette, dann sind lediglich in den Zeilen 60, 80, 120 und 180 die DOPEN und DCLOSE gegen OPEN und CLOSE mit den entsprechenden Gerätenummern auszutauschen. Nun sind wir eigentlich schon fertig: In Zeile 100 löschen wir alle Variablen, den Grafikbildschirm und starten den zweiten Teil des Programmes, der nur zur Kontrolle den File einliest und unser Shape abbildet. Zusätzlich werden auf dem Textbildschirm wieder die Stringlänge und die eingelesenen ASCII-Werte abgebildet. Beide Ausdrucke lassen sich gut vergleichen (Benutzer eines Bildschirmes kommen in diesen Genuß, wenn sie nach Ende des Programmes durch GRAPHIC0 den Textbildschirm einschalten).

c. Eine unorthodoxe Lösung

Etwas exotisch und daher hier nur kurz skizziert ist die folgende Vorgehensweise: Mittels des POINTER-Befehls (hier also POINTER(A$)) sucht man in BANK 1 den Stringdescriptor. Die dort genannte Adresse und Länge verhilft uns dazu, den Speicherbereich, in dem wir unseren String stehen haben, durch BSAVE abzuspeichern.

Beim Wiedereinladen müßte man zunächst einen String definieren, dann durch BLOAD dessen Inhalt an eine beliebige Speicherstelle packen und den Stringdescriptor darauf richten.

Das erscheint im ersten Moment ziemlich kompliziert zu sein, könnte aber – besonders bei Verwendung vieler Shapes – schneller funktionieren. Damit würde beispielsweise ein festgelegter Speicherbereich für alle Shapes en bloc ein- oder auslesbar werden. Probieren Sie's doch mal!

3. Shapes für ein Listing

a. Als Zeichenvorschrift

Häufig werden Shapes durch einige wenige Grafik-Befehle erstellt. Die einfachste Methode – der wir uns auch bisher bedient haben in den Beispielprogrammen – ist es daher, die Zeichenvorschrift ins Programm einzuarbeiten. Manchmal stößt man damit aber an Grenzen: Wenn beispielsweise der Ausschnitt eines Multicolorbildes als Shape verwendet werden soll, das auf dem Grafiktablett erstellt wurde oder wenn es sich um ein Teil eines Fractals handelt, das 25 Stunden bis zur Fertigstellung dauert etc.

b. Shapes in DATA-Zeilen

Ähnlich wie wir bei Sprites einen DATA-Generator verwendet haben, der sich nach seiner Verwendung selbsttätig löscht, geschieht das nun hier auch. Im Programm »SHAPE DATA« (Listing 12) ist außerdem noch ein kleiner Testteil enthalten.

Wie gehabt, wird hier zuerst ein Shape (ein Kreis) erzeugt, dann endet das Programm an einem STOP in Zeile 6. Wenn Sie mit dem 40-Zeichen-Bildschirm arbeiten, dann fügen Sie bitte vor das STOP-Kommando noch ein GRAPHIC 0 ein, denn alles, was nun folgt, ist Text. Das STOP-Kommando wurde eingebaut, um Ihnen die Möglichkeit zu geben, an dem Shape vor dem Aufruf des DATA-Generators noch etwas zu verändern. Ist alles in Ordnung, dann geben Sie bitte CONT ein. Der DATA-Generator erzeugt nun acht Zeilen mit DATAs und dazu noch das Einleseunterprogramm sowie die Löschanweisung DELETE 63000-63012.

Die erste Zahl in den DATAs ist die Stringlänge, die als A später beim Auslesen und Zusammenfügen des Shape-Strings eine wichtige Rolle spielt. Ist das Programm mit dem Ausdrucken der neuen Zeilen fertig, dann bricht es ab. Durch die HOME-Taste gelangen Sie in die erste Zeile. Mittels mehrfach wiederholtem RETURN können nun alle neuen Zeilen ins Programm übernommen und der Generator gelöscht werden. Hinterher sieht unser Programm dann aus wie in »SHAPE DATA 2« (Listing 13). Zur Kontrolle lassen Sie doch mal dieses Programm durch RUN 10 starten. RUN löscht ja auch die Strings, so daß das nunmehr gezeichnete Shape aus den DATA-Zeilen wiederauferstanden ist.

Die Vorgehensweise bei diesem Programm ist dieselbe wie beim DATA-Generator für die Sprites. Allerdings werden hier die Werte nicht aus dem Speicher, sondern aus einem String geholt, der deshalb auch schon durch einen – zumindest teilweisen – Programmlauf definiert worden sein muß.

Wenn Sie mehrere Shapes auf diese Weise festhalten wollen, ist das Programm relativ einfach umzubauen, am besten mittels eines String-Arrays und einer weiteren Schleife.

Wenden wir uns nun dem Zusammenspiel von Sprites und Shapes zu:

Shapes und Sprites

Erzeugen von Sprites aus Shapes.

Anstatt mit dem SPRDEF-Befehl können Sprites auch regelrecht gezeichnet werden durch Grafik-Befehle im Hochauflösungs- oder im Mehrfarbenmodus. Die Nahtstelle zwischen Sprite und Shape ist dabei der String, in den unsere Zeichnung zuerst als ein Shape (mittels SSHAPE) eingeschrieben wird. Von dort ist es dann durch SPRSAV in den Sprite-Speicher zu übertragen.

Eine Regel ist dabei zu beachten:

Der Sprite-String liegt in seiner Länge fest. Im Normalmodus müssen wir immer von einem 24 mal 21 (das ist X mal Y) Raster ausgehen, im Mehrfarbenmodus vom 12 mal 21 Muster. Eine SSHAPE-Anweisung muß daher - falls wir ein Sprite konstruieren wollen - im Normalmodus lauten:

SSHAPE A$,X,Y,X+23,Y+20

und im Multicolormodus:

SSHAPE A$,X,Y,X+11,Y+20

X und Y sollen die Koordinaten der linken oberen Ecke des gewünschten Bildausschnittes sein.

Ist SSHAPE nicht in diesem Raster erstellt worden, kann sich allerlei Unsinn auf dem Bildschirm abspielen.

Ein kleines Demoprogramm (Listing 13) namens »SHAPE UND SPRITE« spielt diese Option durch. Zunächst wird ein Multicolorshape erzeugt und in Sprite 1 geschrieben. Eine Umstellung vom Multicolor- in den Hochauflösungsmodus zeigt, daß das Sprite seine Farben behält, das Shape dagegen nicht. Ein zweites Shape erzeugt Sprite 2, die beide über den Bildschirm bewegt werden. Falls Sie nach Programmende mittels GRAPHIC 0 noch auf den Textbildschirm (40 Zeichen) umschalten, sehen Sie noch einen Unterschied zwischen Shapes und Sprites: Letztere bleiben auch hier aktiv.

Shapes aus Sprites bauen

Natürlich ist auch der umgekehrte Weg möglich: Nämlich Shapes aus Sprites zu konstruieren. Fügen Sie einfach mal an unser letztes Programm aus Listing 13 »SHAPE UND SPRITE« folgende Zeilen an:

130 CLR :REM LOESCHEN DER STRINGS
150 SPRSAV 1,A$:SPRSAV 2,B$
160 GSHAPE A$,150,100:GSHAPE B$,200,100

Über den Sinngehalt solch einer Zuordnung läßt sich natürlich streiten. Denkbar wäre es, das anzuwenden bei Speicherung eines Shape-Musters im Spritespeicher und anschließendem Zurücklesen dieser Information. Darüber hatten wir oben bei Aufbewahrung von Shapes für die Nachwelt schon kurz nachgedacht.

Wenn Shapes und Sprites zusammenstoßen

Kollisionen von Sprites mit Shapes werden vom Computer ebenso registriert, als wären es Zusammenstöße mit Textbestandteilen. Sowohl der COLLISION- als auch der BUMP Befehl können verwendet werden. In einer erweiterten Version unseres Programmes »SHAPE UND SPRITE« namens »SHAPE/SPR-KOLL« (Listing 14) werden alle Möglichkeiten, die wir in den letzten Abschnitten besprochen haben, ausprobiert. Dabei findet der COLLISION-Befehl Anwendung und Sprite 1 ist durch einen Joystick in Port 2 steuerbar. Bei jedem Zusammenstoß wechselt das Sprite die Farben.

Wann Shapes und wann Sprites?

Jeder von Ihnen kann eigentlich nun die Frage beantworten, wann welches der beiden Grafik-Objekte sinnvoll einzusetzen ist: Shapes und Sprites.

- Bewegte Objekte sollten Sprites sein
- Objekte, die in verschiedenen Grafikmodi (auch im 40-Zeichen-Textmodus) in gleicher Gestalt auftreten sollen, können nur Sprites sein.
- Statische Objekte in beliebiger Anzahl sind als Shapes gut zu verwirklichen
- Objekte, bei denen kein starr festgelegter Raster zugrundegelegt wird, können Shapes sein.

Nun sind Ihrer Phantasie keine Grenzen gesetzt: Bauen Sie sich ein Menü, in dem mit einem Sprite in Pfeil- oder Handform auf Symbole (Shapes) gedeutet wird, oder basteln Sie sich Schaltpläne aus vorgefertigten Einzelteilen (Shapes) zusammen, die Sie mit einem Kran-Sprite aussuchen und plazieren oder, oder.

(H.Ponnath/og)

```
10 rem ***** data-lader fuer alle sprites *****
15 fast
20 gosub100:bsave"sprites1",onb0,p3584top4095
30 gosub100:bsave"sprites2",onb0,p3584top4095
40 gosub100:bsave"scene1",onb0,p3584top4095
50 gosub100:bsave"scene2",onb0,p3584top4095
60 gosub100:bsave"scene3",onb0,p3584top4095
65 slow
70 printchr$(147)"alle sprite-files sind gespeiche
rt"
80 end
100 rem *** daten in sprite-speicher schreiben ***
110 fori=3584to4095:reada:pokei,a:nexti
120 return
63000 rem ******* sprite-daten *********
63010 rem *** datas von file sprites1 ***
63020 data000,000,000,000,000,000,000,170,000,001,
254,000,002,001,000,004,000,12
8,004,204,128,004,000,128,060,048,240,036,000,144,
036,132
63021 data144,036,120,144,034,001,016,001,254,000,
000,132,000,000,132,000,000,13
2,000,000,132,000,000,132,000,003,135,000,000,000,
000,000
63022 data000,000,000,000,000,000,000,170,000,001,
254,000,002,001,000,004,000,12
8,004,204,128,004,000,128,060,048,224,036,000,160,
036,132
63023 data160,036,120,160,034,001,032,001,254,032,
000,132,032,000,130,000,000,12
9,000,000,128,128,000,128,096,003,128,000,000,000,
000,000
63024 data000,000,000,000,000,000,000,170,000,001,
254,000,002,001,000,004,000,12
8,132,204,128,132,000,128,252,048,224,004,000,160,
004,132
63025 data160,004,120,160,002,001,032,001,254,034,
000,132,034,000,067,254,000,06
4,000,000,064,000,000,064,000,001,192,000,000,000,
000,000
63026 data000,000,000,000,000,000,000,170,000,001,
254,000,002,001,008,004,000,13
6,004,204,136,132,000,136,252,048,248,132,000,136,
004,132
63027 data136,004,120,142,002,001,000,001,254,000,
000,128,000,000,064,000,000,03
2,000,000,032,000,000,032,000,000,224,000,000,000,
000,000
63028 data000,000,000,000,000,192,000,170,096,001,
254,048,002,001,024,004,000,14
0,132,204,139,132,000,136,252,048,248,004,000,128,
004,132
63029 data128,004,120,128,002,001,000,001,254,000,
000,128,000,000,064,000,000,03
2,000,000,032,000,000,032,000,000,224,000,000,000,
000,000
63030 data000,000,000,000,000,000,255,170,000,161,
254,000,162,001,000,036,000,19
2,036,204,160,036,000,144,060,048,248,004,000,128,
004,132
63031 data128,004,120,128,002,001,000,001,254,000,
000,128,000,000,064,000,000,03
2,000,000,032,000,000,032,000,000,224,000,000,000,
000,000
63032 data000,000,000,000,000,000,000,170,000,001,
254,000,002,001,000,004,000,19
2,004,204,160,004,000,144,060,048,248,036,000,128,
036,132
63033 data128,164,120,128,098,001,000,033,254,000,
016,128,000,008,064,000,004,03
2,000,002,032,000,004,032,000,008,224,000,000,000,
000,000
63034 data000,000,000,000,000,000,000,170,000,001,
254,000,002,001,000,004,000,12
8,004,204,128,004,000,128,028,048,224,020,000,160,
020,132
```



```
63035 data160,020,120,160,018,001,032,017,254,032,
016,132,032,031,252,032,000,03
9,224,000,036,000,000,036,000,000,231,000,000,000,
000,000
63039 rem *** datas von file sprites2 ***
63040 data085,085,085,106,170,169,111,255,249,111,
255,249,111,255,249,109,085,12
1,109,170,121,109,190,121,109,190,121,109,190,121,
109,190
63041 data121,109,190,121,109,190,121,109,190,121,
109,170,121,109,085,121,111,25
5,249,111,255,249,111,255,249,106,170,169,085,085,
085,000
63042 data255,255,255,148,081,069,170,170,139,213,
021,085,162,138,041,213,085,08
1,168,170,169,148,081,069,170,170,163,197,085,085,
162,138
63043 data041,149,085,021,138,170,171,148,081,069,
170,168,171,209,085,085,162,13
8,041,213,069,085,170,042,171,148,017,069,255,255,
255,000
63044 data000,000,001,000,000,003,000,000,007,000,
000,013,000,000,025,000,000,04
9,000,000,097,000,000,193,000,001,129,000,003,001,
000,006
63045 data001,000,012,001,000,024,129,000,048,129,
000,096,129,000,192,129,001,12
8,001,003,000,001,006,000,001,012,000,001,031,255,
255,000
63046 data128,000,000,192,000,000,224,000,000,176,
000,000,152,000,000,140,000,00
0,134,000,000,131,000,000,129,128,000,128,192,000,
128,096
63047 data000,148,048,000,148,024,000,148,012,000,
148,006,000,148,003,000,128,00
1,128,128,000,192,128,000,096,128,000,048,255,255,
248,000
63048 data031,255,255,012,000,001,006,000,001,003,
002,161,001,130,161,000,194,16
1,000,098,161,000,050,161,000,024,001,000,012,001,
000,006
63049 data001,000,003,001,000,001,129,000,000,193,
000,000,097,000,000,049,000,00
0,025,000,000,013,000,000,007,000,000,003,000,000,
001,000
63050 data255,255,248,128,000,048,128,000,096,168,
128,192,168,129,128,165,003,00
0,165,006,000,162,012,000,128,024,000,128,048,000,
128,096
63051 data000,128,192,000,129,128,000,131,000,000,
134,000,000,140,000,000,152,00
0,000,176,000,000,224,000,000,192,000,000,128,000,
000,000
63052 data125,245,087,109,181,086,109,182,166,109,
182,246,109,182,246,109,182,24
6,109,182,246,109,182,246,109,182,246,085,181,086,
085,181
63053 data086,105,182,170,109,182,255,109,182,255,
109,182,255,109,182,255,109,18
2,255,109,182,255,109,182,255,109,182,255,174,186,
255,000
63054 data095,255,214,103,255,102,105,253,166,110,
118,182,111,154,246,111,235,24
6,111,255,246,111,255,246,111,219,250,111,219,255,
111,085
63055 data085,111,154,106,111,219,111,111,235,111,
111,255,111,111,255,111,111,25
5,111,111,255,111,111,255,111,111,255,111,111,255,
111,000
63059 rem *** datas von file scene1 ***
63060 data170,170,170,170,170,170,170,170,170,170,
170,170,165,085,090,165,085,09
0,165,085,090,165,085,090,165,255,090,165,255,090,
165,255
63061 data090,165,255,090,165,255,090,165,255,090,
165,255,090,165,085,090,165,08
5,090,165,085,090,165,085,090,170,170,170,170,170,
170,000
63062 data170,170,170,170,170,170,154,170,166,154,
170,166,165,085,090,165,085,09
0,167,085,218,167,085,218,165,255,090,165,255,090,
165,235
63063 data090,165,235,090,165,235,090,165,255,090,
165,255,090,167,085,218,167,08
5,218,165,085,090,165,085,090,154,170,166,154,170,
166,000
63064 data106,170,169,106,170,169,154,170,166,154,
170,166,173,085,122,173,085,12
2,167,085,218,167,085,218,165,190,090,165,190,090,
165,235
63065 data090,165,235,090,165,235,090,165,190,090,
165,190,090,167,085,218,167,08
5,218,173,085,122,173,085,122,154,170,166,090,170,
165,000
63066 data106,170,169,106,170,169,154,170,166,149,
085,086,173,085,122,173,085,12
2,167,085,218,167,255,218,165,190,090,165,170,090,
165,235
63067 data090,165,195,090,165,235,090,165,170,090,
165,190,090,167,255,218,167,08
5,218,173,085,122,173,085,122,149,085,086,090,170,
165,000
63068 data106,170,169,106,170,169,149,085,086,149,
085,086,157,085,118,157,085,11
8,151,255,214,151,255,214,151,170,214,151,170,214,
151,130
63069 data214,151,130,214,151,130,214,151,170,214,
151,170,214,151,255,214,151,25
5,214,157,085,118,157,085,118,149,085,086,090,170,
165,000
63070 data106,170,169,106,170,169,181,085,094,181,
085,094,157,085,118,158,255,18
2,150,255,150,151,170,214,151,040,214,151,040,214,
151,130
63071 data214,151,130,214,151,130,214,151,040,214,
151,040,214,151,170,214,150,25
5,150,158,255,182,157,085,118,181,085,094,090,170,
165,000
63072 data106,170,169,085,085,085,181,085,094,181,
085,094,159,255,246,158,255,18
2,150,170,150,151,170,214,151,040,214,151,000,214,
151,130
63073 data214,151,130,214,151,130,214,151,040,214,
151,040,214,151,170,214,150,25
5,150,158,255,182,157,085,118,181,085,094,090,170,
165,000
63074 data085,085,085,085,085,085,117,085,093,127,
255,253,095,255,245,094,170,18
1,094,170,181,094,000,181,094,000,181,094,020,181,
094,020
63075 data181,094,020,181,094,000,181,094,000,181,
094,170,181,094,170,181,095,25
5,245,127,255,253,117,085,093,085,085,085,085,085,
085,000
63079 rem *** datas von file scene2 ***
63080 data213,085,087,213,085,087,127,255,253,123,
255,237,123,255,237,124,170,06
1,124,170,061,126,065,189,126,065,189,126,020,189,
126,020
63081 data189,126,020,189,126,065,189,126,065,189,
124,170,061,124,170,061,123,25
5,237,123,255,237,127,255,253,213,085,087,213,085,
087,000
63082 data213,085,087,239,255,251,111,255,249,123,
255,237,120,170,045,124,170,06
1,124,065,061,126,065,189,126,085,189,126,020,189,
126,020
63083 data189,126,020,189,126,085,189,126,065,189,
124,065,061,124,170,061,120,17
0,045,123,255,237,111,255,249,239,255,251,213,085,
087,000
63084 data255,255,255,239,255,251,239,255,251,250,
170,175,250,170,175,248,000,04
7,248,000,047,248,085,047,248,085,047,248,125,047,
248,125
63085 data047,248,125,047,248,085,047,248,085,047,
248,000,047,248,000,047,250,17
0,175,250,170,175,239,255,251,239,255,251,255,255,
255,000
63086 data191,255,254,239,255,251,239,255,251,242,
170,143,242,170,143,249,000,11
1,249,000,111,248,215,047,248,215,047,248,125,047,
248,125
63087 data047,248,125,047,248,215,047,248,215,047,
249,000,111,249,000,111,242,17
0,143,242,170,143,239,255,251,239,255,251,191,255,
254,000
63088 data191,255,254,239,255,251,234,170,171,242,
170,143,240,000,015,249,000,11
1,249,085,111,248,215,047,248,255,047,248,125,047,
248,125
63089 data047,248,125,047,248,255,047,248,215,047,
249,085,111,249,000,111,240,00
0,015,242,170,143,234,170,171,239,255,251,191,255,
```

Listing 1. Alle »ALLE SPRITES«
Dieser DATA-Lader erzeugt auf der Diskette die in den Beispielprogrammen benötigten Sprite-Files.

```
254,000
63090 data191,255,254,234,170,171,234,170,171,224,
000,011,224,000,011,225,085,07
5,225,085,075,225,255,075,225,255,075,225,255,075,
225,255
63091 data075,225,255,075,225,255,075,225,255,075,
225,085,075,225,085,075,224,00
0,011,224,000,011,234,170,171,234,170,171,191,255,
254,000
63092 data191,255,254,202,170,163,202,170,163,228,
000,027,228,000,027,227,085,20
3,227,085,203,225,255,075,225,255,075,225,235,075,
225,235
63093 data075,225,235,075,225,255,075,225,255,075,
227,085,203,227,085,203,228,00
0,027,228,000,027,202,170,163,202,170,163,191,255,
254,000
63094 data170,170,170,138,170,162,128,000,002,164,
000,026,165,085,090,163,085,20
2,163,255,202,161,255,074,161,235,074,161,235,074,
161,235
63095 data074,161,235,074,161,235,074,161,255,074,
163,255,202,163,085,202,165,08
5,090,164,000,026,128,000,002,138,170,162,170,170,
170,000
63099 rem *** datas von file scene3 ***
63100 data170,170,170,128,000,002,128,000,002,133,
085,082,133,085,082,135,255,21
0,135,255,210,135,170,210,135,170,210,135,170,210,
135,170
63101 data210,135,170,210,135,170,210,135,170,210,
135,255,210,135,255,210,133,08
5,082,133,085,082,128,000,002,128,000,002,170,170,
170,000
63102 data128,000,002,133,085,082,135,255,210,135,
170,210,135,170,210,135,170,21
0,135,170,210,135,170,210,135,170,210,135,170,210,
135,170
63103 data210,135,170,210,135,170,210,135,170,210,
135,170,210,135,170,210,135,17
0,210,135,170,210,135,255,210,133,085,082,128,000,
002,000
63104 data000,000,000,021,085,084,031,255,244,030,
170,180,030,170,180,030,170,18
0,030,170,180,030,170,180,030,170,180,030,170,180,
030,170
63105 data180,030,170,180,030,170,180,030,170,180,
030,170,180,030,170,180,030,17
0,180,030,170,180,031,255,244,021,085,084,000,000,
000,000
63106 data085,085,085,127,255,253,122,170,173,122,
170,173,122,170,173,122,170,17
3,122,170,173,122,170,173,122,170,173,122,170,173,
122,150
63107 data173,122,170,173,122,170,173,122,170,173,
122,170,173,122,170,173,122,17
0,173,122,170,173,122,170,173,127,255,253,085,085,
085,000
63108 data085,085,085,111,255,249,122,170,173,122,
170,173,122,170,173,122,170,17
3,122,170,173,122,170,173,122,150,173,122,150,173,
122,085
63109 data173,122,150,173,122,150,173,122,170,173,
122,170,173,122,170,173,122,17
0,173,122,170,173,122,170,173,111,255,249,085,085,
085,000
63110 data169,085,106,175,255,250,191,170,254,190,
170,190,186,170,174,122,170,17
3,122,150,173,122,150,173,122,150,173,122,085,173,
121,125
63111 data109,122,085,173,122,150,173,122,150,173,
122,150,173,122,170,173,186,17
0,174,190,170,190,191,170,254,175,255,250,169,085,
106,000
63112 data170,170,170,175,255,250,191,255,254,191,
255,254,190,150,190,190,150,19
0,190,150,190,190,150,190,189,085,126,189,125,126,
189,255
63113 data126,189,125,126,189,085,126,190,150,190,
190,150,190,190,150,190,190,15
0,190,191,255,254,191,255,254,175,255,250,170,170,
170,000
63114 data170,170,170,175,255,250,191,255,254,191,
255,254,191,085,254,189,085,12
6,189,125,126,189,255,126,189,195,126,189,195,126,
189,195
63115 data126,189,195,126,189,255,126,189,125,126,
189,085,126,189,085,126,191,08
5,254,191,255,254,191,255,254,175,255,250,170,170,
170,000
```


Listing 1. »Alle Sprites« (Schluß)

```
10 rem ***** testprogramm 1 - collision *****
20 bload"sprites2",onb0
30 sprcolor3,6
40 movspr7,300,100:movspr8,300,200
50 sprite7,1,2,0,0,0,1:sprite8,1,1,1,1,1,1
60 collision1,90
70 movspr8,+0,+5
80 goto70
90 collision1
100 print"kollision erfolgt"
110 collision1,130
120 return
130 collision1
140 print"noch eine kollision"
150 collision1,130
160 return
```

Listing 2. Kollision von Sprites

```
10 rem ***** testprogramm 2 bump-befehl *****
20 bload"sprites2",onb0
30 sprcolor3,6
40 movspr7,100,100:movspr8,150,200
50 sprite7,1,2,0,0,0,1:sprite8,1,1,1,1,1,1
60 v=bump(1):if v>0 then gosub85
70 movspr8,+1,+3
80 goto60
85 print:print"bump(1) ergibt den wert"v
90 fori=0to7:a=2↑i:if a and v then print"sprite "i
+1"kollidiert
100 nexti
110 return
```

Listing 3. Ein Programm zur BUMP-Abfrage

```
10 rem ***** testprogramm 3  joy-befehl *****
20 bload"sprites2",onb0
30 sprcolor3,6
40 movspr1,100,100:movspr8,200,200
50 sprite1,1,2,0,0,0,1:sprite8,1,1,1,1,1,1
60 collision1,110
70 x=200:y=200
80 onjoy(2)gosub190,200,210,220,230,240,250,260
90 movspr8,x,y
100 goto80
110 collision1
120 print"kollision erfolgt"
130 collision1,150
140 return
150 collision1
160 print"noch eine kollision"
170 collision1,150
180 return
190 x=x:y=y-1:return
200 x=x+1:y=y-1:return
210 x=x+1:y=y:return
220 x=x+1:y=y+1:return
230 x=x:y=y+1:return
240 x=x-1:y=y+1:return
250 x=x-1:y=y:return
260 x=x-1:y=y-1:return
```

Listing 4. Steuern mittels JOY

```
10 rem ***** vergleich spritedaten *****
20 rem in spritespeicher und im string
30 dima$(7)
40 bload"sprites1",onb0
50 forj=0to7
60 :printchr$(147)chr$(18)j+1".sprite"chr$(146):pr
int:sprsavj+1,a$(j)
70 :bank0:print"datenspeicher:"
80 :fori=0to63
90 ::print(peek(3584+64*j+i));
100 :nexti:print:print"string:"
110 :fork=1tolen(a$(j))
120 ::printasc(mid$(a$(j),k,1));
130 :nextk:print:print"bitte taste druecken"
140 :getkeyb$
150 :print
160 nextj
```

Listing 5. »VERGLEICH SPRDT«. Programm zum Vergleichen von Spritemustern im Speicher und im String



```
62999 rem *** datas aus spritespeicher ***
63000 fast:bank0:forj=0to7
63001 a$(j)="":fori=0to62:a$=str$(peek(3584+64*j+i
))
63002 a$=right$(a$,len(a$)-1):a$="000"+a$:a$=right
$(a$,3)
63003 a$(j)=a$(j)+a$+",":nexti
63004 a$(j)=a$(j)+"000"
63005 nextj:slow
63006 forj=0to7:printchr$(147);
63007 print63020+2*j"data"left$(a$(j),127)
63008 print63021+2*j"data"right$(a$(j),127):print"
goto63009":stop
63009 nextj:printchr$(147);
63010 print"63040 fori=3584to4095:reada:pokei,a:ne
xti:return"
63011 print"delete 63000-63011":stop
```

Listing 6. »SPRITEDATAS«. Erzeugen von DATA-Zeilen aus Spritedaten

```
10 rem ***** vier sprites *****
20 goto110
30 y=y-3:return
40 x=x+3:y=y-3:return
50 x=x+3:return
60 x=x+3:y=y+3:return
70 y=y+3:return
80 x=x-3:y=y+3:return
90 x=x-3:return
100 x=x-3:y=y-3:return
110 bload"sprites2",onb0
120 printchr$(147):color0,1:color4,1:color5,8
130 x=50:y=60
140 movspr3,x,y:movspr4,x+125,y:movspr5,x,y+122:mo
vspr6,x+125,y+122
150 sprite3,1,8,1:sprite4,1,8,1:sprite5,1,8,1:spri
te6,1,8,1
160 do until a=101
170 :movspr4,-1,y:movspr5,x,-1:movspr6,-1,-1
180 :a=a+1
190 loop
200 bank15:sys65520,10,20:a$="joystick in port2"
210 do until joy(2)=128
220 :color5,3
230 :gosub330
240 :color5,1
250 :gosub330
260 loop:x=rsppos(3,0):y=rsppos(3,1)
270 color5,8:printchr$(147):fori=1to50:next
280 do
290 :onjoy(2)gosub30,40,50,60,70,80,90,100
300 :movspr3,x,y:movspr4,x+24,y:movspr5,x,y+21:mov
spr6,x+24,y+21
310 loop until joy(2)=128
320 end
330 sys65520,10,20:printa$:fork=1to50:nextk:return
```

Listing 7. »VIER SPRITES«. Demonstration zum Koppeln von Sprites

```
10 rem ***** ein sprite-trick *****
20 bload"sprites1",onb0
30 printchr$(147)
40 for i=1 to 8:movspri,200,100:nexti
50 for i=1 to 10:for j=1 to 8
60 if j=1 then sprite8,0:else spritej-1,0
70 spritej,1,i+1,1,1,1,0
80 for k=1 to 25:nextk:nextj
90 nexti
100 sprite8,0:sprite1,0
```

Listing 8. »SPRITE-TRICK«. Ein kleiner Trickfilm

```
10 rem ***** sshape tabelle *****
20 fast:open1,4:print#1,"y-diff.","x-diff.","strin
glaenge"
30 deffnh(x)=int((abs(x)+1)/8+.99)*(abs(y)+1)+4
40 y=0
50 do until y=200
60 :y=y+1:x=0
70 :do until x=320
80 ::x=x+1
90 ::z=fnh(x):ifz>255thenprint#1,y1,x1,z1:exit
100 ::x1=x:y1=y:z1=z
110 :loop
120 loop
130 close1:slow:end
```

Listing 9. »SSHAPE-TABELLE«. Eine Hilfe für Shapes: Eine Tabelle wird gedruckt

```
10 rem *** die gshape-modi ***
20 rem erstellen eines shape
30 color0,1:color1,6:color4,1:graphic1,1
40 circle1,10,10,10,10:box1,0,0,20,20:paint1,1,1:p
aint1,19,1:paint1,1,19
50 paint1,19,19:circle1,10,10,7,7:draw1,10,0to10,2
0:draw1,0,10to20,10
60 sshapea$,0,0,20,20
70 scnclr1
80 rem bildschirmhintergrund
90 dx=5:dy=5:x=0:y=0
100 do while x<511
110 :x=x+dx:y=y+dy
120 :draw1,0,ytox,0
130 loop
140 rem abbilden des shape
150 fori=0to4
160 :gshapea$,10+55*i,100,i
170 nexti
180 rem kommentar
190 color1,3
200 fori=0to4
210 :b$=right$(str$(i),1)
220 :char1,2+7*i,18,b$
230 nexti
240 char1,1,4,"die gshape-modi und ihre wirkungen:
",1
250 char1,1,23,"bitte eine taste"
260 getkeya$
270 graphic0,1
280 end
```

Listing 10. »GSHAPE-MODI«. Die 5 Shape-Darstellungsformen

```
10 rem***** shape erstellen *****
20 color0,1:color4,1:color1,6:graphic1,1:b$=chr$(1
3)
30 circle1,10,10,10,10:sshapea$,0,0,20,20:gosub230
40 scratch"shape1"
50 rem***** shapes auf diskette speichern *****
60 dopen#1,"shape1",w:print#1
70 fori=1tolen(a$):print#1,asc(mid$(a$,i,1))b$:nex
ti:print#1,"ende"b$
80 dclose#1
90 rem***** variable loeschen und neustart *****
100 clr:graphic1,1:sleep2:run120
110 rem***** shapes wieder von diskette holen ****
*
120 dopen#1,"shape1"
130 do
140 :input#1,b$
150 ::if b$="ende" then exit
160 :b=val(b$):b$=chr$(b):a$=a$+b$
170 loop
180 dclose#1:gosub230
190 rem***** shape zeichnen *****
200 gshapea$,100,100
210 end
220 rem***** kontrollunterprogramm *****
230 fori=1tolen(a$):printasc(mid$(a$,i,1));:nexti:
print:printlen(a$):return
```

Listing 11. »SHAPE SPEICHERN«. Shapes retten auf Diskette oder Kassette

```
1 rem ****** erzeugung von data-zeilen aus shape-s
trings (hier a$) ******
2 rem
3 rem ****** probeshape erzeugen ******
4 rem
5 color0,1:color4,1:color1,6:graphic1,1:circle1,10
,10,10,10:sshapea$,0,0,20,20
6 stop
7 rem ****** erzeugen der datazeilen ******
8 goto63000
9 rem ****** neues programm zur kontrolle der data
s ******
10 color0,1:color4,1:color1,3:graphic1,1:gosub6301
5
```

Listing 12. »SHAPE DATA«. Shapes in DATA-Zeilen ablegen ***(Fortsetzung nächste Seite)***

```
15 gshapec$,100,100
20 end
62999 rem ****** es folgt der datagenerator bzw di
e datas ******
63000 printchr$(147)63020"data"len(a$)
63001 i=1:k=0
63002 do
63003 :z=0:print63021+k"data";
63004 :do while i<len(a$)
63005 ::printasc(mid$(a$,i,1))",";:i=i+1:z=z+1
63006 ::ifz=10thenexit
63007 :loop
63008 :printasc(mid$(a$,i,1)):i=i+1:k=k+1
63009 loop while i<len(a$)
63010 print63021+k"data"asc(mid$(a$,i,1))
63011 print"63015 reada:fori=1toa:readb:b$=chr$(b)
:c$=c$+b$:nexti:return"
63012 print"delete63000-63012"
```

Listing 12. »SHAPE DATA« (Schluß)

```
10 rem ***** shapes zu sprites umwandeln *****
15 rem********** multicolor-sprites **********
20 color0,1:color1,6:color2,2:color3,3:color4,1:co
lor5,2
30 graphic3,1:circle1,5,10,5,10:box2,0,0,10,20:dra
w3,0,10to10,10
40 draw3,5,0to5,20
50 sshapea$,0,0,11,20
60 sprsava$,1:sprcolor6,3:movspr1,100,100:sprite1,
1,2,0,0,0,1
65 sleep2
70 rem********** normaler hiressprite **********
80 graphic1,1:circle1,50,10,10,10:circle1,50,10,7,
7:circle1,50,10,4,4
90 sshapeb$,40,0,63,20:gshapea$,0,0
100 sprsavb$,2:movspr2,200,200:sprite2,1,6
110 movspr1,0#4:movspr2,90#8
```

Listing 13. »SHAPE UND SPRITE«. Shapes werden zu Sprites

```
10 rem ***** kollision von sprites und shapes ****
*
20 rem ***** shapes zu sprites umwandeln *****
30 rem********** multicolor-sprites **********
40 color0,1:color1,6:color2,2:color3,3:color4,1:co
lor5,2
50 graphic3,1:circle1,5,10,5,10:box2,0,0,10,20:dra
w3,0,10to10,10
60 draw3,5,0to5,20
70 sshapea$,0,0,11,20
80 sprsava$,1:sprcolor6,3:movspr1.100,100:sprite1,
1,2,0,0,0,1
90 sleep2
100 rem********** normaler hiressprite **********
110 graphic1,1:circle1,50,10,10,10:circle1,50,10,7
,7:circle1,50,10,4,4
120 sshapeb$,40,0,63,20:gshapea$,0,0
130 sprsavb$,2:movspr2,200,200:sprite2,1,6
140 movspr1,0#4:movspr2,90#8
150 sleep2
160 rem********** loeschen der strings **********
170 clr
180 rem********** uebertragen von sprites in shape
s **********
190 sprsav1,a$:sprsav2,b$
200 gshapea$,150,100:gshapeb$,200,100
210 rem********** steuern von sprite 1 **********
220 char1,20,0,"joystick port 2"
230 rem********** kollision m. shapes **********
240 movspr1,0#0
250 x=rsppos(1,0):y=rsppos(1,1):i=6:j=3
260 collision2,370
270 onjoy(2)gosub290,300,310,320,330,340,350,360
280 goto270
290 y=y-2:movspr1,x,y:return
300 x=x+1:y=y-1:movspr1,x,y:return
310 x=x+2:movspr1,x,y:return
320 x=x+1:y=y+1:movspr1,x,y:return
330 y=y+2:movspr1,x,y:return
340 x=x-1:y=y+1:movspr1,x,y:return
350 x=x-2:movspr1,x,y:return
360 x=x-1:y=y-1:movspr1,x,y:return
370 collision2
380 i=i+1:ifi>16theni=1
390 j=j+3:ifj>16thenj=1
400 sprcolori,j
410 collision2,370
420 return
```

Listing 14. »SHAPE/SPR-KOLL«. Zusammenspiel von Sprites und Shapes

Apfelmännchen: Schönheit im Chaos

Fractals: Ausdruck einer Revolution im naturwissenschaftlichen Weltbild, sichtbar gemacht auf dem Bildschirm. Was es damit auf sich hat, wie sie entstehen und wie man sie auf dem C128 erzeugen kann, zeigt Ihnen dieser Artikel.

Als Beispiel für die Anwendung von Mehrfarbengrafik auf unserem C128 behandeln wir ein Thema, das in den letzten Jahren zunehmend an Bedeutung gewonnen hat: die sogenannten Fractals. Durch Rekursion (dieser Begriff wird gleich noch erklärt) meist sehr einfacher mathematischer Beziehungen lassen sich mit Hilfe des Computers Bilder von überraschender Schönheit erzeugen. Der Computer und die neuesten Erkenntnisse der angewandten Mathematik als Instrumente der bildenden Kunst? Versuchen Sie es selbst!

Die Realität ist fractal!

Bevor wir uns die Praxis ansehen – Rezepte dafür sind häufig zu finden [1][6] – sollten wir uns etwas mit der Bedeutung der Fractals befassen. Daß solch ein Gebilde nämlich als Computerkunst unsere ästhetische Ader anspricht, ist eigentlich nicht mehr als ein erfreulicher Nebeneffekt. In Wahrheit sind diese hübschen Bilder ein Ausdruck des derzeitigen Wandels im naturwissenschaftlichen Weltbild, der von vielen Zeitgenossen geradezu als Revolution empfunden wird. Das erfordert eine Erklärung:

Prinzipiell existieren in der Natur zwei gegenläufige Tendenzen: eine zerstreuende (dissipative) und eine ordnende. Letztere findet ihren Ausdruck beispielsweise in der Regelmäßigkeit von Kristallen oder Planetensystemen. Die klassische Physik versucht seit Jahrtausenden

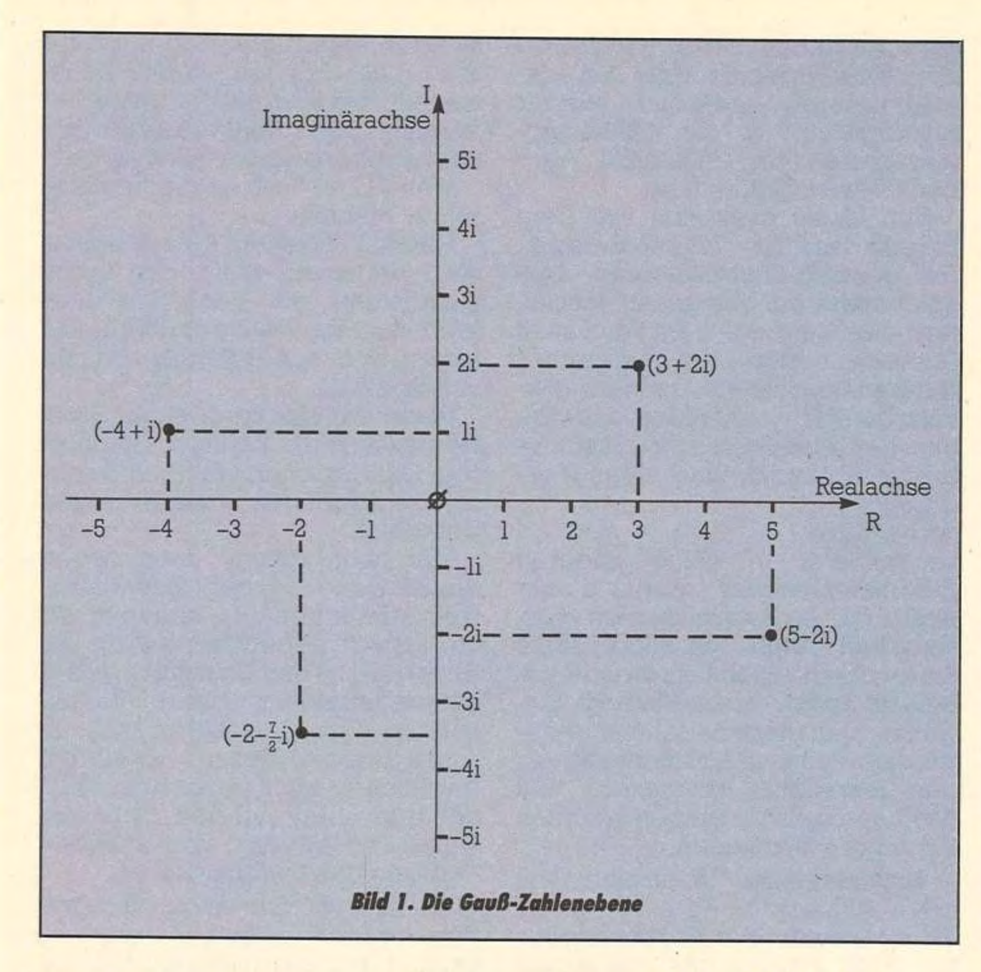
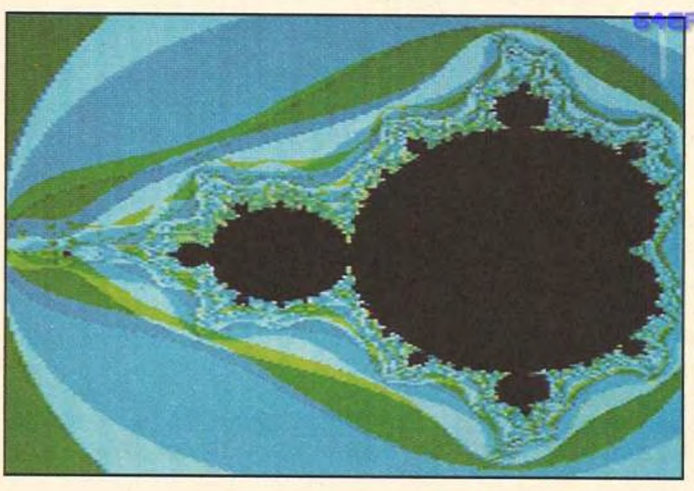


Bild 1. Die Gauß-Zahlenebene


Bild 2. Die normale Mandelbrot-Menge: das »Apfelmännchen«

alles Weltgeschehen von diesem Denkansatz (die Ordnung der Dinge zu ergründen) her zu erklären. Durch Idealisieren und Abstrahieren wurden Theorie und Experiment zum mächtigen Gebäude der Naturwissenschaften zusammengefügt.

Relativ neu im Vergleich dazu ist die Untersuchung der anderen – der zerstreuenden – Tendenz. Sie findet ihren ersten Ausdruck in der sogenannten Molekularkinetik (Stichworte dazu sind Entropie und Wärmetod) und ihren derzeitigen Höhepunkt in der modernen Quantentheorie. Hier arbeitet der Naturwissenschaftler mit dem Instrumentarium der Statistik und Wahrscheinlichkeitsrechnung. Vielerorts herrscht aber die Meinung, daß man – wenn die Materie bis in die grundlegenden Bausteine erkannt sei – prinzipiell auch diese Bereiche der Natur mit den Mitteln einer erweiterten klassischen Physik beschreiben kann (das Wort »klassisch« ist hier etwas gewagt, weil zu dieser Art Physik auch die Erkenntnisse der Relativitätstheorie gehören würden).

Eine Basis des naturwissenschaftlichen Denkens ist der sogenannte Determinismus: Jede Wirkung hat ihre Ursache. Der Glaube, daß man – kennt man nur alle Einflußgrößen und physikalischen Zusammenhänge und verfügt man über ausreichende Rechenkapazität – jedes Ereignis vorherberechnen kann, ist so alt wie unsere Kultur. Daher der Gebrauch des Wortes »klassisch«, denn auch die Relativitätstheorie bricht nicht mit dem Glauben an Ursache-Wirkungs-Ketten. Diese Basis ist in jüngster Zeit erschüttert worden durch die Erkenntnisse zweier Arbeitsgebiete der modernen Naturwissenschaft, die auf den ersten Blick nicht viel miteinander zu tun haben.

Der Begriff der »Ursache« einer Wirkung wird in der modernen Elementarteilchenphysik (in der S-Matrix-Theorie) in Frage gestellt [5]. Von der anderen Seite sägt B.Mandelbrot am Fundament des Determinismus: Er arbeitet am Thema »Extreme und unvorhersagbare Unregelmäßigkeiten von Naturphänomenen in Physik, Soziologie und Biologie«. Gleichzeitig ein Mittel zur Beschreibung und ein Ausdruck solcher Unregelmäßigkeiten in der Natur sind die Fractals. Mit der Verleihung der Barnard-Medaille 1985 durch die Nationale Akademie der Wissenschaften der Vereinigten Staaten ist die Bedeutung seiner Arbeiten für die Wissenschaft deutlich geworden: Frühere Preisträger sind beispielsweise Albert Einstein, Niels Bohr und Werner Heisenberg [4].

Was sind Fractals?

Drei Ansätze zum Verständnis von Fractals werden wir nun kennenlernen: Einen geometrischen, einen, der mit der Dynamik von Wachstumsprozessen zusammenhängt und schließlich noch einen mathematischen Ansatz.

Fractals und Geometrie. Mandelbrot hat einmal erklärt: »Die Natur hat der klassischen Geometrie ein Schnippchen geschlagen. Denn Wolken sind eben keine Kugeln, Berge keine Kegel, Inseln keine Kreise und Baumstämme keine Zylinder. Ebenso bewegt sich ein Blitz nicht auf einer Geraden. Und die Oberfläche der Planeten ist nicht glatt.« (Zitiert in [4]). Es ergibt sich die Notwendigkeit, eine neue Geometrie anzuwenden, eine Geometrie der Natur.

Objekte dieser Geometrie werden durch rekursive Methoden (wir kommen wirklich bald zur Erklärung, was das eigentlich ist) gewonnen. Sie haben merkwürdige Eigenschaften. So gibt es Kurven, die eine kleine Fläche umranden

und dennoch unendlich lang sind. Die Unendlichkeit drückt sich darin aus, daß man bei immer genaueren Hinsehen immer kompliziertere Strukturen erkennt. Der Begriff der Dimension (Erinnern Sie sich: Ein Punkt ist ein Gebilde mit 0 Dimensionen- er hat keine Ausdehnung. Eine Linie gehört zur ersten Dimension, ihre Ausdehnung läßt sich als Länge angeben. Eine Fläche ist zweidimensional, weil wir von Länge und Breite reden können. Körper sind dreidimensionale Objekte, bei denen noch die Höhe eine Rolle spielt.), dieser Begriff der Dimensionen also gerät ins Wanken: Die Objekte der neuen Geometrie sind beispielsweise nicht mehr eindimensional, aber noch nicht zweidimensional (oder aber nicht mehr zwei- und noch nicht dreidimensional). Sie haben eine gebrochene, nichtganzzahlige Dimensionalität. Daher der Name Fractal, vom lateinischen »frangere«, was brechen heißt.

Insekten und Fractals

Ein Beispiel für fractale Geometrie nennt Fricker [4]: Ein britisches Forschungsteam hat die Oberfläche von gewissen Pflanzen untersucht und dafür die (fractale) Dimension 2.79 bestimmt. Die Folgerung daraus klärt einen Widerspruch: Werden diese Pflanzen von zehnfach kleineren Insekten als ursprünglich bevölkert, dann nimmt deren Zahl nicht - wie man erwarten sollte - um das Hundertfache, sondern um etwa das Sechshundertfache zu. Den Grund dafür erklärt die Rechnung mit Fractals: $10^{2.79} = 617$, anstelle von $10^2 = 100$.

Fractals und Wachstumsdynamik. Die Gesetze des exponentiellen Wachstums sind schon seit langem bekannt. Am Beispiel der Bevölkerungsexplosion sollen sie kurz erklärt werden. Wenn X(n) die Bevölkerung im Jahr n ist und p die jährliche Zuwachsrate, dann ergibt sich für die Bevölkerung im Jahr n+1 die Beziehung:

X(n+1) = (1+p)*X(n)

Will man nun die Bevölkerungszahl im Jahr n+2 erfahren, dann setzt man in diese Gleichung anstelle von X(n) nun X(n+1) ein. Für das Jahr n+3 setzt man X(n+2) ein und so fort. Es ergibt sich eine immer steiler steigende Kurve, wenn man die Bevölkerungszahlen X(i) gegen die jeweiligen Jahreszahlen i grafisch darstellt: die Bevölkerungsexplosion.

Was wir bei diesem Beispiel gemacht haben, nennt man Rekursion. Das Ergebnis einer Berechnung wurde jedesmal wieder zurückgeführt in die Gleichung. »recurrere« ist lateinisch und bedeutet »zurücklaufen«.

Nun ist es nicht erst seit dem Bericht des Club of Rome bekannt, daß gewisse Einflußfaktoren – beispielsweise ein Mangel an Rohstoffen oder Nahrung – dem Wachstum Grenzen setzen. Ein belgischer Naturwissenschaftler namens Verhulst hat schon in der Mitte des vorigen Jahrhunderts ein Wachstumsgesetz entdeckt, das einen Ausdruck

p*(1-a*X(n))

verwendete. In völlig anderen Zusammenhängen spielt dieser Ausdruck in der modernen Wissenschaft eine wichtige Rolle (Laserphysik, Evolutionstheorie, turbulente Strömung von Medien). Entwickelt man diese Beziehung rekursiv, dann kann – in Abhängigkeit von den jeweiligen Parametern und nach genügend häufigen Schritten – dreierlei geschehen:

- Exponentielles Wachstum (wie schon gehabt)
- Zurückgehen der Ergebnisse bis auf Null oder einen jeweiligen Grenzwert
- Dazwischen aber gibt es merkwürdige Verhaltensweisen. Man kann bei bestimmten Ausgangswerten ein Oszillieren der Ergebnisse beobachten, das mehr oder weniger regelmäßig erfolgt [3] und stark von dem exakten Wert der Wachstumsrate abhängt. Schon eine klitzekleine Änderung der Ausgangsbedingungen kann entscheiden über einen völlig anderen Verlauf der Wachstumskurven.

Dieses Grenzdasein ist untrennbar mit Fractals verbunden. Auf welche Weise, das soll uns der nächste Abschnitt zeigen. Interessant in dem Zusammenhang ist es, daß die bisher erwähnten Anspielungen auf Fractals ebenfalls immer Grenzen betrafen: Die Oberfläche einer Pflanze, die Umrandung einer Fläche, die schmale Linie zwischen exponentiellem Wachstum einerseits und Abfallen auf Null andererseits...

Ein einfacher mathematischer Weg zu Fractals: B. Mandelbrot [9,10] hat eine Anzahl Funktionen untersucht, von denen eine der vorhin gezeigten Wachstumsfunktion ähnelt:

$X(n+1) = X(n)^2 + C$

Wir nehmen also einen beliebigen Wert X(n), quadrieren ihn und addieren einen konstanten Wert C. Wieder setzen wir das Ergebnis X(n+1) als X(n) neu ein und beobachten, wie sich bei fortgeführter Iteration (also weiteren Rekursionen) der berechnete Wert verhält.

Wenn C = 0 ist, ist die Situation relativ einfach:

Falls wir mit einem X-Wert kleiner als 1 beginnen, ergibt sich durch Quadrieren ein noch kleinerer Wert und bei weiterer Rekursion nähert sich das Ergebnis schließlich der Null.

Wenn wir dagegen einen Startwert größer als 1 wählen, dann ist sein Quadrat noch größer und sehr schnell strebt das Ergebnis gegen Unendlich.

Man nennt diese Zahlen, gegen die jeweils die Ergebnisse tendieren »Attraktoren« [3]. So haben wir im bisher betrachteten Fall die Attraktoren 0 und Unendlich. Jeder dieser Attraktoren besitzt Einflußsphären: Zu 0 gehören alle X-Anfangswerte, deren Absolutbetrag kleiner als 1 ist, zu Unendlich gehören alle anderen Anfangswerte. Die Zahlen 1 und -1 bilden Grenzen der Einflußsphären.

Wählen wir nun einen anderen Wert für die Konstante C, beispielsweise C=1. Beginnt man nun mit X=0, dann erhält man der Reihe nach:

0, 1, 2, 5, 26, 677,...

Welchen Startwert für X wir auch immer wählen (probieren Sie es mal aus!), jedesmal entwickelt sich das Ergebnis nach wenigen Schritten zur Unendlichkeit hin. Ein Attraktor ist verlorengegangen!

Nehmen wir nun mal den Wert -1 für die Konstante C und starten mit X=1, dann finden wir diese Ergebnisreihe:

1, 0, -1, 0,- 1, 0, -1, 0, ...

Die Werte pendeln hin und her um einen periodischen Attraktor, der seinen Einflußbereich auf die X-Startwerte zwischen -1.618 und +1.618 ausdehnt. Versuchen wir beispielsweise den Startwert .5, dann ergibt sich:

.5, -.75, -.4375, -.8086, -.346, -.880, -.2253, etc.

Alle anderen Startwerte tendieren schnell zur Unendlichkeit: Beispielsweise beim Startwert 3:

3, 8, 63, 3968, ...

Wir haben demnach nun wieder 2 Attraktoren, von denen einer ein periodischer ist und der andere die Unendlichkeit.

Probieren Sie dieses Spiel noch etwas durch mit verschiedenen Konstanten C und Startwerten X. Sie werden immer finden, daß ein Attraktor unendlich ist. Für C-Werte

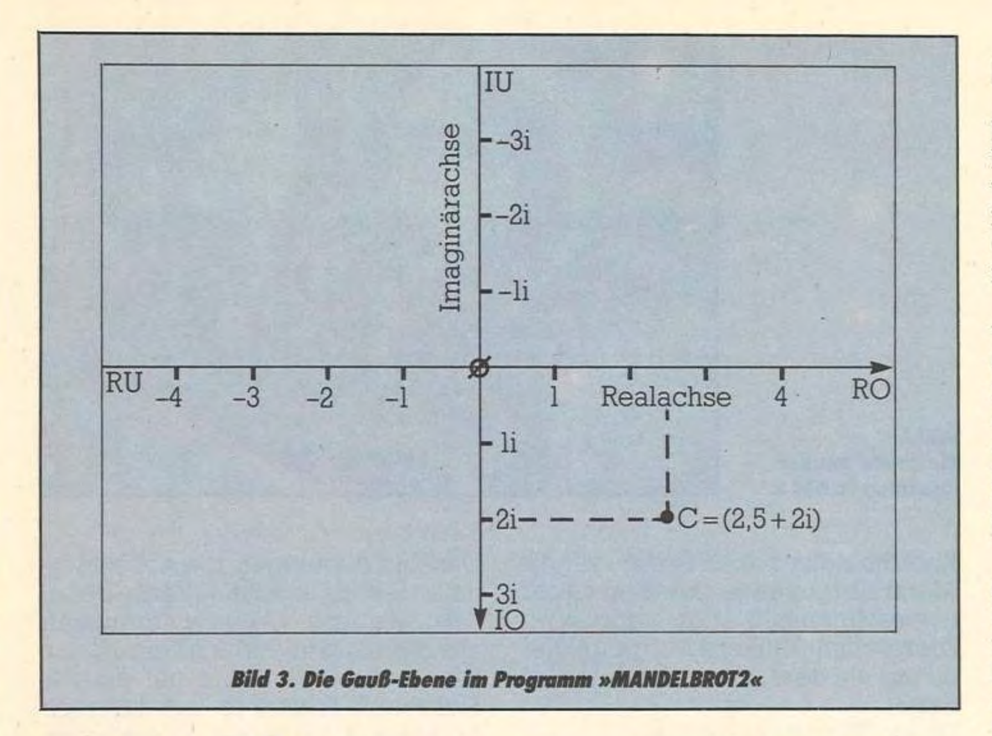


Bild 3. Die Gauß-Ebene im Programm »MANDELBROT2«

zwischen -2 und 0.25 gibt es dann noch einen 2. Attraktor, der entweder ein einfacher oder ein periodischer ist oder aber ein merkwürdig chaotischer [3].

Wir sehen schon, daß mit dieser einfachen mathematischen Gleichung allerhand Interessantes anzustellen ist. Um nun zur grafischen Darstellung dieser Zusammenhänge zu gelangen – denn das sind die Fractals, wie Sie sie in Abbildungen sehen –, werden wir noch eine besondere Art von Zahlen kennenlernen, die uns wesentlich mehr Möglichkeiten bietet.

Komplexe Zahlen

Keine Angst, es wird jetzt nicht plötzlich abgehoben in die höheren Sphären der Mathematik. Komplexe Zahlen sind nämlich im Grunde genommen ganz einfach zu verstehen. Ihre Einführung ergab sich aus der Wurzelrechnung. Eine Quadratwurzel wie $\sqrt{4}$ ist ja definiert als die Zahl, die mit sich selbst multipliziert das Argument der Wurzel ergibt. In unserem Beispiel ist das Argument die 4 und 2*2 ist 4, somit ist die Quadratwurzel von 4 eine 2. Weitere Beispiele:

$3 = \sqrt{9}$

$1 = \sqrt{1}$

aber auch $-3 = \sqrt{9}$, weil auch $(-3)*(-3) = 9$ gilt. Probleme werden Sie haben, wenn Sie auf Ihrem Computer mal versuchen, die Aufgabe PRINT SQR(-2) zu lösen. Es gibt keine Zahl aus dem normalen Zahlenbereich, die mit sich selbst malgenommen eine negative Zahl ergibt. Der Computer steigt mit einer Fehlermeldung aus. Auch die Menschen sind bei dieser Fragestellung lange Zeit »ausgestiegen«. Irgendwann (das war im 17. Jahrhundert) kam aber mal jemand auf die Idee, eine neue Sorte von Zahlen einzuführen , die sogenannten imaginären Zahlen, deren einfachstes Glied die Zahl i ist. Dabei sollte $i=\sqrt{-1}$ sein. Man kann tatsächlich damit rechnen und manches Mal ergeben sich aus solchen Rechnungen wieder unsere gewohnten Zahlen, wenn nämlich einmal i^2 auftritt, welches -1 ist:

$i = \sqrt{-1}$

$i^2 = -1$

Noch etwas komplizierter wird das mit den sogenannten komplexen Zahlen. Die sind zusammengesetzt aus einem Realteil (das sind unsere normalen Zahlen) und einem Imaginärteil (also einem Teil, in dem i eine Rolle spielt). Solche Zahlen sehen immer so aus:

$z = a + b*i$

Realteil Imaginärteil

Mit solchen komplexen Zahlen kann man auch wunderbar rechnen, was Ihnen an der Addition gezeigt werden soll:

$(2+3i) + (4-i) = 6 + 2i$

Sie sehen, man behandelt einfach beide Teile gesondert $(2+4=6)$, berechnet die Summe des Realteils und $(3i-i=2i)$ die des Imaginärteils). Alle Rechenarten sind jedenfalls möglich, nur wir werden sie hier nicht zeigen.

Zwei Begriffe spielen für unsere weiteren Absichten noch eine Rolle. Zunächst dreht es sich dabei um den Betrag einer komplexen Zahl. Wenn wir von der komplexen Zahl $z = a + bi$ ausgehen, dann ist $|z| = SQR(a^2 + b^2)$ ihr Betrag.

Zum zweiten soll noch eine Methode zur grafischen Darstellung komplexer Zahlen vorgestellt werden, die der berühmte Mathematiker Gauß entwickelt hat: die Gauß'sche Zahlenebene. In Bild 1 ist das Prinzip erläutert:

Auf den ersten Blick sieht das aus wie ein normales Koordinatensystem. Bei näheren Hinsehen erkennen Sie, daß in der horizontalen Achse der Realteil, in der vertikalen aber der Imaginärteil einer komplexen Zahl erfaßt wird. Jeder komplexen Zahl entspricht nun ein Punkt auf dieser Ebene.

Noch eine kleine Bemerkung zur sicher bei Ihnen auftretenden Frage, was man denn um Himmels willen mit diesen merkwürdigen Zahlen anfangen kann. Fragen Sie doch mal einen Ingenieur oder einen Elektrotechniker oder einen Physiker etc.

Damit haben wir nun fast alles erforderliche Rüstzeug, um Fractals erzeugen zu können.

Die Mandelbrot-Menge

Erinnern wir uns an die Gleichungen, die wir vorhin für eine mathematische Einführung in die Fractals gebraucht haben. Auch hier verwenden wir nun solch eine Beziehung:

$m = z^2 + c$

Hier ist allerdings z eine komplexe Variable (wir verwenden im folgenden dafür $z=x+yi$) und c eine komplexe Konstante (im folgenden ist dann $c=a+bi$).

Ausführlich geschrieben haben wir dann die Gleichung:

$m = (x+yi)^2 + (a+bi)$

Auch hier arbeiten wir rekursiv, setzen also jedes Ergebnis wieder als neues z in die Gleichung ein.

Wir beginnen mit $z=0$, wobei dann x und y gleich 0 sind. Setzt man das in die Gleichung ein, dann erhält man: $m1 = z^2 + c = c$

Der zweite Schritt setzt anstelle von z nun c ein: $m2 = c^2 + c$

Nun wird die Sache interessant. Im 3. Schritt folgt nämlich:

$m3 = (c^2+c)^2 + c$

Sie sehen jetzt zweierlei: Zum einen wird die Angelegenheit im weiteren Verlauf schnell unübersehbar und zum zweiten können Sie sich sicher vorstellen, daß – je nach dem Wert von c – schnell große Werte als Ergebnis auftreten werden. Jedes Ergebnis m kann man als Punkt auf der Gauß-Ebene darstellen. Die meisten Zahlen für c

allerdings werden bald zu Ergebnissen führen, die aus jedem im noch so großen Maßstab gezeichneten Gauß-System heraus ihren Weg zur Unendlichkeit antreten.

Probieren wir das mal mit einem c aus (2): c = 1 + 1i. In der Reihenfolge der Ergebnisse finden sich dann:

m1 =	1	+	1i
m2 =	1	+	3i
m3 =	-7	+	7i
m4 =	1	-	97i
m5 =	-9407	-	193i
m6 =	88454401	+	3631103i

Allerdings steigen nicht bei allen c-Werten die Ergebnisse über alle Grenzen. Einige existieren, bei denen auch beliebig häufige Iteration immer noch zu endlichen Zahlen führt. Die Menge dieser C-Werte nennt man die Mandelbrot-Menge, nach ihrem Entdecker B. Mandelbrot. Das ist die Figur, die in der grafischen Darstellung das »Apfelmännchen« ergibt.

Wenn also der Realteil von C etwa zwischen -2 und +0.5, der Imaginärteil etwa zwischen -1.25 und +1.25 liegt, dann hat man gute Chancen, auch nach extrem vielen Iterations-Schritten noch endliche Ergebnisse zu erhalten, denn C liegt dann innerhalb der Mandelbrotmenge.

Ist C so gewählt, daß es außerhalb dieses Bereiches darzustellen ist (die Darstellungsebene ist eine Gauß-Zahlenebene!), dann streben die Ergebnisse schon nach mehr oder weniger Iterationen gegen Unendlich.

Die Trennzone aber zwischen diesen beiden Möglichkeiten hat fractalen Charakter. Dazu werden wir gleich noch kommen. Wir wollen uns nun die ganze Materie unter programmtechnischen Gesichtspunkten ansehen.

Programm zum Abschätzen der Mandelbrotmenge

Dewdney [2] erwähnt, daß die Iterationstheorie uns einen untrüglichen Maßstab in die Hand gibt, der uns erkennen läßt, wann eine Rekursion zum Attraktor Unendlich führt. Mit Hilfe des Betrages unserer Ergebnisse ist das möglich: Wenn nämlich dieser Betrag irgendwann einmal größer als 2 wird, dann treibt bald danach auch das Ergebnis gegen Unendlich.

Entwickeln wir also zunächst ein kleines Programm, das uns die Ergebnisse der Iterationen und ihren Betrag – das ist nämlich die Entfernung vom Nullpunkt des Koordinatensystems, in dem wir die Mandelbrotmenge gesehen haben – berechnen hilft. Dazu bringen wir zuerst einmal unsere Ausgangsgleichung in eine computergerechte Form:

Diese Gleichung hieß ja:

$m = z^2 + c$

und dabei waren die Zahlen z und c komplex:

$z = x + yi$

$c = a + bi$

Ausgeschrieben hieße dann unsere Gleichung:

$m = (x+yi)^2 + (a+bi)$

Daraus folgt durch Umformung:

$m = (x^2 - y^2 + a) + (2xy + b)i$

Realteil Imaginärteil

Wir definieren nun zwei Funktionen. Für die Berechnung des jeweiligen Realteils ist das

FN R(V) = X*X - Y*Y + A

und für die Berechnung des Imaginärteils

FN I(V) = 2*X*Y + B

Jede neue Iteration erfordert nun nur noch die Einsetzung des Ergebnisses in z, also die Zuweisungen:

R = FN R(V):X = R

I = FN I(V):Y = I

Dabei ist übrigens V lediglich eine Dummy-Variable. Zur Berechnung des Betrages brauchen wir noch die Beziehung:

D = SQR(R↑2 + I↑2)

Das Programm »MANDELBROT1« (Listing 1) übernimmt all diese Aufgaben und druckt Ihnen auf dem Bildschirm folgende Angaben aus: Iterationszahl, Real- und Imaginärteil des jeweiligen Ergebnisses und schließlich den Betrag. Probieren Sie nun mal eine Reihe von Kombinationen durch. Für den Anfang raten wir Ihnen, für X und Y 0 einzusetzen. Sie bewegen sich dann nämlich im System der Mandelbrotmenge, wie es Bild 2 zeigt.

Beim Ausprobieren finden Sie Punkte für C, die sehr schnell zu einem Betrag größer als 2 führen. Das sind diejenigen, die sich außerhalb des »Apfelmännchens« befinden. Andere C-Werte brauchen anscheinend unendlich lange dazu. Hier haben wir dann mit einiger Sicherheit Elemente der Mandelbrotmenge vorliegen. Dazwischen aber liegen Bereiche für C, bei denen der Betrag lange Zeit braucht, bis er größer als 2 wird. Das ist wieder ein Grenzbereich, der fractalen Charakter aufweist. Wie fractaler Charakter aussehen kann, wird uns nun das nächste Programm zeigen.

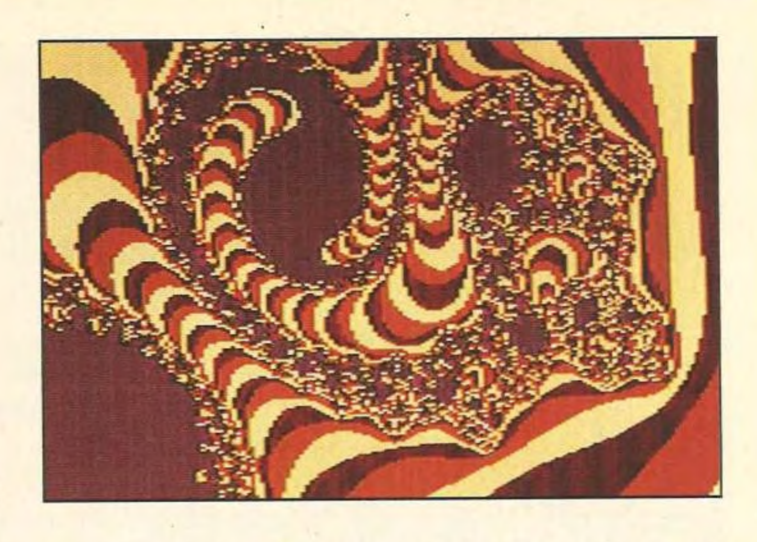


Bild 5. Ein Fractal aus der Randzone in Bild 2.

Das Zeichnen von Fractals

Das Ausprobieren mit dem Programm »MANDELBROT1» könnte uns auch der Computer abnehmen. Wir schreiben ein Programm »MANDELBROT2« (siehe Listing 2), das alle Kombinationen für C (also die Kombinationen für den Realteil und den Imaginärteil) innerhalb gegebener Grenzen selbst ausführt.

Dazu brauchen wir vier Eingaben, nämlich die Unter- und die Obergrenzen des Realteils sowie des Imaginärteils, innerhalb derer C variiert werden soll. Dann müssen wir uns eine sinnvolle Form der Ergebnisausgabe überlegen. Wir wählen – wie Sie sich nun schon denken können – eine grafische Ausgabe, die unsere Multicolorgrafikfähigkeit ausnutzt.

Auf dem Bildschirm wird ein Koordinatensystem definiert, dessen Y-Achse die Imaginärachse einer Gauß-Ebene ist (und vom kleinsten zum größten Imaginärteil von C reicht). Die X-Achse soll die Variation des Realteils von C ausdrücken und geht daher vom vorher eingegebenen unteren zum oberen Realteil von C. Alle Punkte dieser Gaußebene entsprechen dann verschiedenen Konstanten C (siehe Bild 3).





Bild 4. Ein stark verzerrtes »Apfelmännchen«

Das Programm »MANDELBROT2« führt nun für jeden Wert C die Iterationen durch und prüft dabei jedesmal, ob der Betrag des Ergebnisses schon 2 überschreitet. Ist das der Fall, merkt es sich die Anzahl der Schritte bis dahin. Andernfalls folgen weitere Rekursionen bis zu einer vorher angegebenen maximalen Iterationsanzahl. Ist bis zu dieser höchsten Iterationszahl der Betrag immer noch nicht 2, dann wird das dazu gehörige C als schwarzer Punkt eingezeichnet. Dieser Wert C liegt dann in der Mandelbrotmenge. Alle anderen C werden in einer Farbe gezeichnet, die mit der Anzahl der Iterationen zusammenhängt.

Hätten wir bei einer maximalen Iterationszahl von 100 auch 100 verschiedene Farben zur Verfügung, dann wäre unser Problem schon gelöst. Auf diese Weise werden beispielsweise auf größeren Grafik-Computern die Bilder von Fractals erzeugt. Wir sind da leider etwas eingeschränkt auf nur vier Farben, von denen wir eine (zum Beispiel Schwarz) schon für die Punkte aus der Mandelbrotmenge vergeben haben. Um eine sichtbare Struktur zu erzeugen, wiederholen wir in einer bestimmten Reihenfolge die verbleibenden drei Farben. Wir bedienen uns dazu einer Funktion, die leider nicht zum Basic-Wortschatz unseres Basic 7.0 gehört: der modulo-Funktion.

Wie wirkt diese Funktion? Sehen wir uns das am Beispiel von modulo(3) einmal an. Das Ergebnis ist immer der Rest, der bei Division durch 3 übrigbleibt:

Zahl	modulo(3) davon
3	0
4	1
5	2
6	0
7	1 etc.

Wir können uns solch eine modulo-Funktion definieren durch:

DEF FN MD(X) = X - INT(X/D)*D

Hier ist D der Divisor, in unserem Beispiel also die Zahl 3. X ist dann die Zahl, mit der die modulo-Funktion arbeiten soll. In Zeile 4 des Programmes finden Sie diese Funktion, die in Zeile 310 zum Zeichnen benutzt wird. Einige weitere Erklärungen zum Programm:

Die Zeilen 1 bis 4 enthalten alle benötigten Variablen. Sie sind hier so angeordnet, daß die am häufigsten gebrauchten möglichst weit vorne stehen. Der Basic-Interpreter findet sie dann schneller. In 120 bis 160 werden alle Eingaben verlangt:

RU,RO = unterer und oberer Rand des Realteils von C. Innerhalb dieser Grenzen wird dann C variiert.

IU,IO = dasselbe für den Imaginärteil von C.

NMAX = Das ist die maximale Iterationszahl. Man muß bedenken, daß von dieser Zahl die Zeichendauer ganz entscheidend abhängt. Im schlimmsten Fall müssen 160*200*NMAX Iterationen ausgeführt werden, bei NMAX = 100 also schon 3,200,000 Schleifendurchläufe! Das wäre allerdings nur dann der Fall, wenn die gesamte Zeichenebene innerhalb der Mandelbrotmenge läge.

F1,F2,F3,F4 = Das sind die Farben, in denen unsere Abbildung erstrahlen soll. Dabei ist

F1 die Farbe, die mit COLOR3 ins Mehrfarbenregister geschrieben wird,

F2 wird mit COLOR2 in das andere Mehrfarbenregister geschrieben,

F3 landet mittels COLOR1 in dem Register für die aktuelle Vordergrundfarbe und

F4 bestimmt den Rand und die Hintergrundfarbe. In der Farbe F4 wird dann auch die Mandelbrotmenge selbst gezeichnet.

Nach all diesen Eingaben wird in den FAST-Modus geschaltet (sollten Sie zwei Bildschirme verwenden, dann können Sie in den Zeilen 110 und 400 statt GRAPHIC0 ein GRAPHIC5 einsetzen und so den Textbetrieb weiter verfolgen). Zeile 180 berechnet die Schrittweiten der Variation des Real- und des Imaginärteils von C. Es folgt eine Dreifachschleife: Ganz außen wird die Vertikale (also der Imaginärteil) durch die Variable U durchgezählt. Innerhalb dieser Schleife liegt die Schleife für die Horizontale mit dem Schleifenzähler V. Die eigentliche Iteration geschieht in der inneren DO...LOOP-Schleife. Zwei Ausstiegsbedingungen sind vorgegeben: In Zeile 250 wird der Betrag des Ergebnisses als Quadrat (wegen der schnelleren Rechnung) abgefragt. Übersteigt dieser den Wert $2^2=4$, dann ist die Schleife beendet. Die andere Abbruchbedingung befindet sich in Zeile 290, wo die Anzahl der Iterationen verglichen wird mit der maximalen (vorher eingegebenen) Schrittzahl. Je nach dem Ausstieg aus dieser inneren Schleife wird dann in Zeile 310 ein Punkt gezeichnet. Als Farbquelle dient das durch modulo(3) errechnete Register.

Der Rest des Programmes dient zum Abspeichern des errechneten Bildes.

Fractal-Praxis

Das Zeichnen von Fractals ist eine Geduldsprobe: Je nach Anzahl der Iterationen und ausgesuchtem Gauß-Ebenen-Abschnitt müssen Sie mit der Blockade Ihres Computers für zwei bis acht Stunden rechnen! Was wäre da zu tun? Die einzige Lösung ist das Programmieren in Maschinensprache. Weil wir zur Berechnung allerlei Operationen mit Fließkommazahlen brauchen, ist das nur sinnvoll unter Verwendung von Interpreter-Routinen zu programmieren. Die Schöpfer der C128-Firmware sind leider sehr zurückhaltend mit Informationen, so daß ein Programm, das auf diese Interpreter-Routinen zugreift, noch etwas auf sich warten lassen muß. Für den C64-Modus hat G.Pehland in der Zeitschrift 64'er (Ausgabe 11/1985,S.80) ein schönes Programm veröffentlicht. Besitzer des C128 können dieses Programm noch erheblich beschleunigen, indem sie im C64-Modus auf doppelte Geschwindigkeit schalten. Das ist möglich durch folgende POKEs:

POKE 53296,1 = schnell
POKE 53296,0 = normal

Im Programm von G. Pehland müßten diese POKEs wie folgt angeordnet werden:

Zeile 1141 vor dem Befehl SYS CL umschalten auf doppelte Geschwindigkeit und Zeile 1150 nach dem Befehl SYS BG zurückschalten auf normale Geschwindigkeit. Trotz dieser Beschleunigungsmaßnahmen ist auch hier das Zeichnen von Fractals noch eine arge Geduldsprobe.

Pehlands Programm verwendet übrigens einen etwas anderen Algorithmus zum Fractal-Zeichnen, weshalb sein Apfelmännchen umgedreht auf dem Bildschirm erscheint. Eine feine Sache – die Sie auch leicht in unser Programm »MANDELBROT2« einbauen können – ist die Möglichkeit, Ausschnitte eines fertigen Bildes festzulegen für ein weiteres Bild. Pehland verwendet dazu zwei Sprites, die mittels der Cursortasten an die

```
10 rem *** programm mandelbrot 1 ***
20 trap190
30 x=0:y=0:a=0:b=0:n=0:r=0:i=0:d=0:v=0
40 def fn r(v) = x*x-y*y+a
50 def fn i(v) = 2*x*y+b
60 print chr$(147) chr$(17) "mandelbrot-algorithmu
s"
70 print chr$(17) chr$(17) "m = (x+y*i)↑2 + (a+b*i
)" chr$(17)
80 input"a,b";a,b:input"x,y";x,y
90 print chr$(147) "iteration","realteil","imag.te
il","betrag":print
100 r = fn r(v):i = fn i(v)
110 printn,r,i
120 do
130 :x=r:y=i
140 :r = fn r(v):i = fn i(v):n=n+1:d=sqr(r*r+i*i)
150 :printn,r,i,d
160 ::getkeya$:if a$="←" then exit
170 loop
180 end
190 if er = 15 then print"ergebnis ist auf dem weg
 zur unendlichkeit"
200 end
```

Listing 1. »MANDELBROT1«
Ein Programm zum Abschätzen der Mandelbrotmenge.

```
1 n%=0:nm%=0:x=0:y=0:r=0:i=0:d=0:v=0:u=0:a=0:b=0:d
a=0:db=0
2 iu=0:io=0:ru=0:ro=0:f1%=1:f2%=1:f3%=1:f4%=1
3 a$="":b$=""
4 def fn md(x)=x-int(x/3)*3
10 rem ********************************
20 rem *                              *
30 rem *    programm zur grafischen   *
40 rem *        darstellung von       *
50 rem *                              *
60 rem *m a n d e l b r o t m e n g e n*
70 rem *                              *
80 rem *  heimo ponnath  hamburg 1985  *
90 rem *                              *
100 rem********************************
110 color0,1:color4,1:color5,6:graphic3,1:graphic0
,1
120 print:print:print:print chr$(18)"mandelbrotmen
gen als grafik"chr$(146)
130 print:print:print"parameter der gaussebene:":i
nput"ru,ro,iu,io=";ru,ro,iu,io
140 print:print:print"maximale iterationen:":input
"nmax=";nm%
150 print:print:print"farben nach steigendem n:":i
nput"f1,f2,f3,f4=";f1%,f2%,f3%,f4%
160 print:print:print"es wird jetzt eine ganze wei
le dauern!!"
170 sleep5:graphic3:fast:color0,f4%:color4,f4%:col
or1,f3%:color2,f2%:color3,f1%
180 da=(ro-ru)/159:db=(io-iu)/199
190 b=iu-db
200 for u=0 to 199
210 :b=b+db:a=ru-da
220 :for v=0 to 159
230 ::a=a+da
240 ::n%=0:r=0:i=0:d=0
250 ::do while d<4
260 :::x=r:y=i
270 :::r=x*x-y*y+a:i=2*x*y+b:d=r*r+i*i
280 :::n%=n%+1
290 ::::if n%=nm% then exit
300 ::loop
310 ::if n%=nm% then draw0,v,u:else draw(fnmd(n%))
+1,v,u
330 nextv:nextu
340 slow
390 getkeya$:ifa$<>"←"then390
400 graphic0:print:print"bild abspeichern?(j/n)"
410 getkeya$:if a$="j" then begin
420 :input"bildname";b$
430 :bsave(b$),onb0,p7168top16383
440 bend
450 end
```


Listing 2. »MANDELBROT2«
Ein Programm zum Zeichnen von Fractals

linke obere und die rechte untere Ecke des gewünschten Ausschnittes gesteuert werden. Nach Tastendruck rechnet der Computer die Spriteposition um in Koordinaten der Gauß-Ebene.

Ein komplettes »Apfelmännchen« (Bild 2) dient hauptsächlich zur Übersicht. Sie erhalten es durch die Eingaben:
RU = -2
RO = 0.5
IU = -1.25
IO = 1.25

Viel interessanter aber ist der Grenzbereich der (schwarz gezeichneten) Mandelbrotmenge. Je stärker die Vergrößerung des Ausschnittes wird, desto mehr Einzelheiten sind auf dem Bild zu finden. Die Anzahl der Iterationen drückt sich in der Feinheit der Details aus. Je höher sie gewählt wird, desto mehr Einzelheiten zeigt das Bild. Allerdings sind uns durch die karge Auflösung (160 mal 200 Bildpunkte) schnell Grenzen gesetzt. Wählen wir eine zu hohe Anzahl von Iterationen, dann wird das Bild überladen und Informationen werden zugedeckt.

Weniger Iterationen zeigen also oft mehr Information. Die optimale Schrittzahl zu finden, ist eine Sache des Ausprobierens.

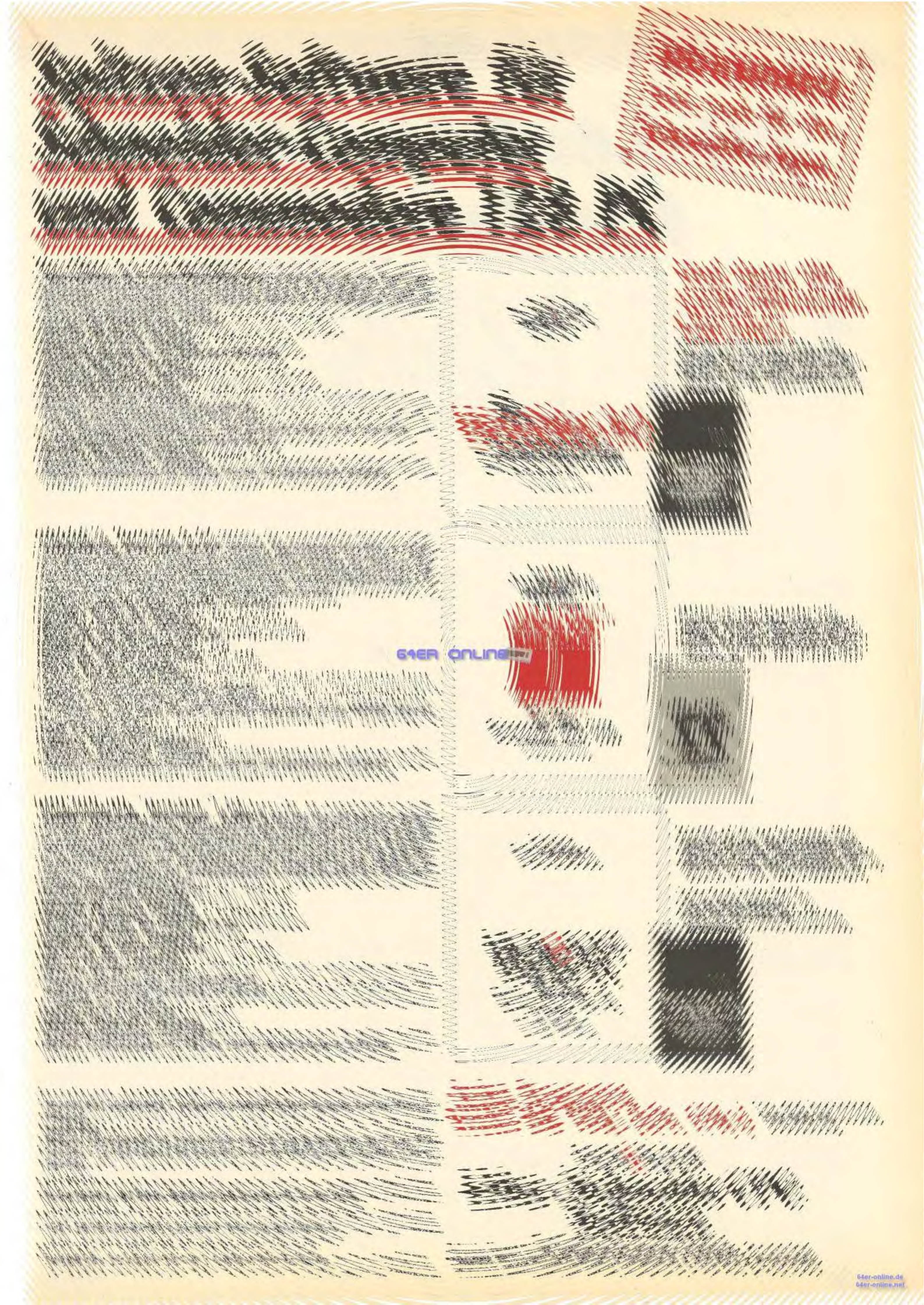
»Schiefe« Mandelbrotmengen ergeben sich, wenn wir uns überlegen, welchen speziellen Algorithmus wir zur Erzeugung gewählt haben und diesen dann etwas verallgemeinern. Im Gegensatz zu »MANDELBROT1«, haben wir in »MANDELBROT2« stillschweigend X und Y (also den Realteil und den Imaginärteil der Variablen in unserer Ausgangsgleichung m = (x + yi)2 + (a + bi)) auf den Wert 0 festgelegt. Wir haben aber schon in »MANDELBROT1« erkennen können, daß sich die Verhältnisse völlig verändern, wenn X und Y andere Werte annehmen. Bauen wir in »MANDELBROT2« noch ein:
135 INPUT "X,Y= ";XR,YI
und ändern Zeile 240 um in
240 N% = 0:R = XR:I = YI:D = 0
dann schaffen wir uns neue Einflußmöglichkeiten. Die »Apfelmännchen«, die wir damit erzeugen, sehen verzerrt aus oder sind häufig gar nicht mehr zu erkennen (Bild 4).

Verwenden Sie doch mal folgende Parameter:
RU = -2;RO = 0.5;IU = -1.25;IO = 1.25
X = 0.1;Y = 0.5
Iterationszahl 100
beziehungsweise die Parameter aus Bild 5:
RU = -.57;RO = 0.7;IU = -1.5;IO = 1
X = -0.6;Y = 0.7
Iterationszahl 100

Die Erforschung der Randgebiete dieser jetzt entstehenden verzerrten Mandelbrotmengen dürfte allerlei interessante neue Bilder ergeben. Und nun viel Spaß beim Ausprobieren neuer Kombinationen, von denen es unendlich viele gibt und einen widerstandsfähigen Computer, der den Rund-um-die-Uhr-Betrieb nicht übelnimmt!

(H.Ponnath/og)

LITERATUR:
[1] G.Pehland:Bilder aus einer anderen Dimension; 64er,Ausg.11(1985)S.80
[2] A.K.Dewdney:Computer-Kurzweil; Spektrum der Wissenschaft, Ausg.10(1985) S.8
[3] H.-O.Peitgen,P.H.Richter: The Beauty of Fractals, Berlin/Heidelberg/New York/ Tokyo 1985, Springer-Verlag (Vorabinformation)
[4] F.Fricker: Barnard-Medaille für Benoit Mandelbrot; Spektrum der Wissenschaft, Ausg.11(1985)S.14
[5] F.Capra: Wendezeit; Bern/München/Wien 1985, Scherz-Verlag
[6] K.Klotz: Computergrafik zum Nachmachen; CHIP, Ausg.10(1984)S.38
[7] M.L.Prueitt: Art and the Computer; New York 1984, Mc-Graw-Hill
[8] J.D.Foley,A. van Dam: Fundamentals of interactive Computer Graphics; Reading.Mass. 1984, Addison-Wesley
[9] B.B.Mandelbrot: Fractals:Form, Chance, and Dimension; San Francisco 1977, Freeman
[10] B.B.Mandelbrot: The Fractal Geometry of Nature; San Francisco, Freeman



80-Zeichen-Grafik für den C 128

Auf dieses Grafikpaket für eine 640 x 200-Punkte-Auflösung auf dem 80-Zeichen-Bildschirm haben C 128-Besitzer gewartet! Alle Grafikbefehle des Basic 7.0 stehen damit für den neuen Modus zur Verfügung.

Der neue Basic-Interpreter des Commodore 128 enthält 26 Befehle, mit deren Hilfe man einfach und schnell auch komplizierte Grafik-Programme erstellen kann. Leider unterstützen diese Befehle nur die Programmierung des VIC-Chips mit der schon vom C 64 her bekannten Auflösung. Der zusätzlich eingebaute 80-Zeichen-Chip »VDC« wird nur zum Aufbau eines 80-Zeichen-Text-Bildschirms genutzt. Wer auch diesen Bildschirm für grafische Darstellungen benutzen will, der muß seine eigenen Routinen schreiben. Doch gibt es – wie der vorliegende Artikel zeigt – eine recht einfache Lösung für dieses Problem. Das abgedruckte Basic-Programm (Listing 1) baut ein Maschinenprogramm für den »User-RAM-Bereich« von $1300 bis $1bff auf. Nach dem Initialisieren dieses Maschinenprogramms mit »SYS DEC ("1303")« stehen die Befehle GRAFIK, BOX, CIRCLE, DRAW und PAINT auch für den VDC-Chip zur Verfügung. Die Befehle LOCATE, SCALE und SCNCLR und die Funktionen RCLR, RDOT und RGR können ohne Einschränkung im 640x200-Grafik-Modus benutzt werden. Beide Bildschirme können gleichzeitig im HiRes-Modus arbeiten, beim Aufruf eines Grafikbefehls prüft der Interpreter selbständig, welcher Video-Chip gerade angesprochen ist.

So programmiert man den 80-Zeichen-Chip

Der 8563-Video-Chip ist ein erweiterter »Abkömmling« des viel verwendeten 6845-CRT-Controllers. Er verfügt (im C 128) über 16 KByte eigenen Bildschirm-RAM und ist nur über zwei Register unter den Adressen 54784 ($d600) und 54785 ($d601) mit den anderen Bausteinen des Computers verbunden. Alle Register und der gesamte Video-RAM-Bereich müssen über diese beiden Adressen angesprochen werden. Das »Einblenden« des Video-RAMs in den Adressenbereich der CPU ist nicht vorgesehen. Will man einen Wert in eines der 36 Register des 8563 übertragen, so muß man zunächst die Nummer des anzusprechenden Registers in die Speicherstelle 54784 und dann den zu übertragenden Wert in die Speicherstelle 54785 schreiben. Genauso geht es beim Lesen eines Registerinhaltes: »POKE 54784, Registernummer: PRINT PEEK (54785): REM Registerinhalt«.

Die HiRes-Grafik-Register

Für die Grafikprogrammierung sind von den 36 Registern des VDC vor allem die Register 18, 19, 31, 25 und 26 interessant. Der Inhalt von Register 26 bestimmt (im Grafikmodus!) Vorder- und Hintergrundfarbe. Über Bit 0 bis 3 kann man 16 verschiedene Hintergrundfarben, über Bit 4 bis 7 16 verschiedene Vordergrundfarben anwählen.

Ein Beispiel: die Befehlsfolge »POKE 54784,26:POKE 54785, (3*16 + 2)« bewirkt, daß im HiRes-Modus eine rote Zeichenfarbe auf weißem Grund erscheint. Im Textmodus wird mit diesem Befehl nur die Hintergrundfarbe verändert. Register 25 ist ein mehrfach belegtes Register. Über die Bits 0 bis 3 kann man den Bildschirm um maximal 16 Pixels horizontal nach links verschieben. Mit Bit 4 kann man zwischen einfacher (0) und doppelter (1) Pixelgröße wählen. Mit den Bits 5 bis 7 kann man zwischen den drei Betriebsarten Text (Bit 6 = 1), Semigrafik (Bit 5 = 1) und Grafik (Bit 7 = 1) wählen.

Für uns ist hier nur interessant, daß man durch Beschreiben von Register 25 mit dem Wert 128 den HiRes-Modus wählen und durch Beschreiben des Registers mit dem Wert 64 wieder in den Text-Modus zurückkehren kann. Alle anderen Bits sind standardmäßig mit Null besetzt. Über die Register 18,19 und 31 wird das Video-RAM angesprochen. Man besetzt in der beschriebenen Weise zunächst Register 18 mit dem High-Byte und Register 19 mit dem Low-Byte der gewünschten Bildschirmspeicheradresse vor.

Durch Schreiben in Register 31 kann man dann einen Wert in die gewählte Speicherstelle schreiben, durch Auslesen von Register 31 erhält man den Inhalt dieser Speicherstelle. Aber Vorsicht! Nach jeder Schreib- oder Leseoperation wird der RAM-Zeiger in Register 18/19 automatisch einen Schritt erhöht! Will man also einen Wert aus dem Video-RAM auslesen und anschließend (verändert) wieder zurückschreiben, so muß man die Zeiger 18/19 zweimal setzen. Das erscheint zwar sehr umständlich, doch hat diese Arbeitsweise einen überzeugenden Vorteil: 10mal »PEEK (Reg. 31)« ergibt die 10 Bytes von Zeiger 18/19 bis Zeiger 18/19 + 9. Der Zeiger 18/19 muß nur einmal gesetzt werden. Das Auslesen oder Vorbesetzen größerer Speicherbereiche wird so erheblich beschleunigt. Man überlege sich einmal, um wieviel langsamer die Eingabeschleife des Bildschirmeditors würde, wenn Register 18 und 19 auf jedes Zeichen einzeln gesetzt werden müßten.

Organisation des Grafik-Bildschirms

Für jedes Pixel (Grafikpunkt) wird ein Bit benötigt. Das macht acht Pixel pro Byte oder 80 Byte für eine waagerechte Bildzeile bei einer horizontalen Auflösung von 640 Pixels. In vertikaler Richtung erreicht der VDC 200 Zeilen Auflösung, das heißt insgesamt werden 80x200 = 1600 Byte Video-RAM von einem Bild belegt. Ein Farb-RAM ist nicht vorgesehen. Der VDC kann hochauflösende Grafik nur in einer einzigen Farbe darstellen, ein entscheidender Nachteil des VDC gegenüber dem VIC. Auch in der Adressierung der einzelnen Pixels unterscheiden sich VDC und VIC.

```
1 rem "        GRAPHIK-80
2 :
3 rem "erstellt Graphik-Paket für den
4 rem " 80-Zeichen-HIRES-Bildschirm
5 rem "           des  C-128
6 :
7 rem "         Version 1.00
8 :
9 rem "         1985  by  KRW
10 :
11 :
1000 fast
1020 restore
1040 bank 15
1060 scnclr 5: print chr$(17) chr$(17) t
ab(20)"Erstellen eines Graphik-Paketes"
1080 print chr$(17) tab(17)"für den 8563
-HIRES-Graphik-Bildschirm"
1100 print chr$(17) chr$(17) chr$(17) "
 Bearbeitet:"
1120 print: print
1140 :
2000 rem "Erweiterung der Interpretersch
leife laden
2020 :
2040 print tab(25) "Neue Interpreterschl
eife
2060 a= dec("1300"): e= dec("1398")
2080 gosub 8000
2100 :
3000 rem "Neue Routine für 'Punkt setzen
, löschen und testen' laden
3020 :
3040 print tab(25) "Punkt setzen, lösche
n, testen
3060 a= dec("1400"): e= dec("1671")
3080 gosub 8000
3100 :
4000 rem "Erweiterung der GRAPHIC-Routin
e laden
4020 :
4040 print tab(25) "Erweiterung der GRAP
HIK-Routine
4060 a= dec("1a3e"): e= dec("1ab6")
4080 gosub 8000
4100 :
5000 rem "ROM-Routinen in RAM-Bereich ko
pieren
5020 :
5040 print tab(25) "ROM -> RAM - Kopie
5060 a= dec("61a8"): e= dec("6388"): rem
 "PAINT und BOX
5080 g= dec("1672")
5100 gosub 8500
5120 :
5140 a= dec("6797"): e= dec("67d6"): rem
 "DRAW
5160 if g<>dec("1853") then print "Fehle
r !": stop
5180 gosub 8500
5200 :
5220 a= dec("668e"): e= dec("674c"): rem
 "CIRCLE
5240 if g<>dec("1893") then print "Fehle
r !": stop
5260 gosub 8500
5280 :
5300 a= dec("9b30"): e= dec("9c18"): rem
 "Strecke zeichnen
5320 if g<>dec("1952") then print "Fehle
r !": stop
5340 gosub 8500
5360 :
5380 read ad$
5400 do while ad$<>"ende"
5420 read mn$, al$, ah$
5440 ad= dec(ad$)
5460 poke ad, dec(mn$): poke ad+1, dec(a
l$): poke ad+2, dec(ah$)
5480 read ad$
5500 loop
5520 :
5540 print tab(25) "Fertiges Maschinenpr
ogramm abspeichern
5560 if ds >20 then print ds$: stop
5580 bsave "graphik-80.m",d0,u8,on b0,p(
dec("1300")) to p(dec("1ab7"))
5600 if ds >20 then print ds$: stop
5620 sys dec("1303")
5640 print chr$(17) chr$(17) "      Graph
ik-Paket fertig installiert, gespeichert
 und initialisiert."
5660 end
6000 end
7960 rem "Kopierroutine 1
7980 :
8000 for i=a to e
8020 read d
8040 poke i,d
8060 next i
8080 return
8100 :
8460 rem "Kopierroutine 2
8480 :
8500 for i=a to e
8520 poke g, peek(i)
8540 g= g+1
8560 next i
8580 return
8600 :
9000 :
10000 rem "datas für neue Interpretersch
leife
10010 :
10020 data  76,45,19,76,6,19,120,169,45,
141,8,3,169,19,141,9,3,88,169,0,141,0
10030 :
10040 data 255,169,6,141,6,213,169,32,13
3,48,133,50,133,52,169,0,133,47,133
10050 :
10060 data  49,133,51,96,32,128,3,201,22
2,144,31,201,233,176,27,170,36,215,16
10070 :
10080 data  52,189,123,18,141,78,19,189,
135,18,141,79,19,138,32,128,3,32,0,0
10090 :
10100 data  76,246,74,32,134,3,76,243,74
,62,114,215,129,147,141,43,86,88,226
10110 :
10120 data 121,96,26,22,103,23,24,101,10
0,24,105,105,106,105,189,162,18,141
10130 :
10140 data  78,19,189,174,18,141,79,19,7
6,73,19,90,168,215,183,142,141,43,151
10150 :
10160 data  85,226,121,96,107,97,103,98,
```

Listing 1. »Graphik-80« (Basic-Lader). Bitte geben Sie dieses Programm im 40-Zeichen/DIN-Modus ein.




```
102,101,100,103,105,105,106,105,85
10170 :
12000 rem "datas für Routine Punkt setze
n, löschen, testen
12010 :
12020 data  76,48,22,76,9,20,76,93,22,17
3,0,255,72,169,0,141,0,255,169,7,141
12030 :
12040 data  6,213,104,141,0,255,96,173,0
,255,133,158,169,0,141,0,255,133,253
12050 :
12060 data 133,254,173,52,17,208,90,169,
199,205,51,17,144,83,169,127,237,49
12070 :
12080 data 17,169,2,237,50,17,144,71,172
,51,17,185,150,20,133,253,185,99,21
12090 :
12100 data 133,254,174,50,17,173,49,17,4
1,248,74,74,74,24,125,147,20,133,252
12110 :
12120 data 24,165,253,101,252,133,253,16
5,254,105,0,133,254,162,18,32,204,205
12130 :
12140 data 162,19,165,253,32,204,205,32,
216,205,133,252,173,49,17,41,7,170,189
12150 :
12160 data 139,20,24,96,56,96,128,64,32,
16,8,4,2,1,0,32,64,0,80,160,240,64,144
12170 :
12180 data 224,48,128,208,32,112,192,16,
96,176,0,80,160,240,64,144,224,48,128
12190 :
12200 data 208,32,112,192,16,96,176,0,80
,160,240,64,144,224,48,128,208,32,112
12210 :
12220 data 192,16,96,176,0,80,160,240,64
,144,224,48,128,208,32,112,192,16,96
12230 :
12240 data 176,0,80,160,240,64,144,224,4
8,128,208,32,112,192,16,96,176,0,80
12250 :
12260 data 160,240,64,144,224,48,128,208
,32,112,192,16,96,176,0,80,160,240,64
12270 :
12280 data 144,224,48,128,208,32,112,192
,16,96,176,0,80,160,240,64,144,224,48
12290 :
12300 data 128,208,32,112,192,16,96,176,
0,80,160,240,64,144,224,48,128,208,32
12310 :
12320 data 112,192,16,96,176,0,80,160,24
0,64,144,224,48,128,208,32,112,192,16
12330 :
12340 data 96,176,0,80,160,240,64,144,22
4,48,128,208,32,112,192,16,96,176,0
12350 :
12360 data 80,160,240,64,144,224,48,128,
208,32,112,192,16,96,176,0,80,160,240
12370 :
12380 data 64,144,224,48,128,208,32,112,
192,0,0,0,0,1,1,1,2,2,2,3,3,3,4,4,4
12390 :
12400 data 5,5,5,5,6,6,6,7,7,7,8,8,8,9,9
,9,10,10,10,10,11,11,11,12,12,12,13
12410 :
12420 data 13,13,14,14,14,15,15,15,15,16
,16,16,17,17,17,18,18,18,19,19,19,20
12430 :
12440 data 20,20,20,21,21,21,22,22,22,23
,23,23,24,24,24,25,25,25,25,26,26,26
12450 :
12460 data 27,27,27,28,28,28,29,29,29,30
,30,30,30,31,31,31,32,32,32,33,33,33
12470 :
12480 data 34,34,34,35,35,35,35,36,36,36
,37,37,37,38,38,38,39,39,39,40,40,40
12490 :
12500 data40,41,41,41,42,42,42,43,43,43,
44,44,44,45,45,45,45,46,46,46,47,47
12510 :
12520 data47,48,48,48,49,49,49,50,50,50,
50,51,51,51,52,52,52,53,53,53,54,54
12530 :
12540 data54,55,55,55,55,56,56,56,57,57,
57,58,58,58,59,59,59,60,60,60,60,61
12550 :
12560 data61,61,62,62,62,63,63,63,32,28,
20,176,34,166,131,208,5,73,255,37,252
12570 :
12580 data  44,5,252,133,252,162,18,165,
254,32,204,205,162,19,165,253,32,204
12590 :
12600 data 205,162,31,165,252,32,204,205
,165,158,141,0,255,96,32,28,20,176,245
12610 :
12620 data 37,252,240,6,32,87,22,162,0,9
6,32,87,22,162,255,96
12630 :
14000 rem "datas für Erweiterung des GRA
PHIC-Befehls
14010 :
14020 data 201,158,208,11,32,34,160,32,1
28,3,169,0,133,216,96,32,244,135,224
14030 :
14040 data 6,240,41,176,36,138,72,169,0,
141,0,255,162,25,169,64,32,204,205,32
14050 :
14060 data 12,206,165,215,72,169,128,133
,215,32,66,193,104,133,215,104,72,170
14070 :
14080 data  76,110,107,76,40,125,169,0,1
41,0,255,162,25,169,128,32,204,205,32
14090 :
14100 data 28,158,169,0,141,0,255,224,2,
176,229,224,0,240,29,160,64,132,8,160
14110 :
14120 data 0,152,162,18,32,204,205,162,1
9,32,204,205,162,31,32,204,205,136,208
14130 :
14140 data 250,198,8,208,246,96
19900 :
20000 data 1a3b, 4c, 00, 14
20010 data 16d5, 20, 00, 14
20020 data 1a2d, 20, 00, 14
20030 data 188d, 20, 1d, 1a
20040 data 19c7, 20, 1d, 1a
20050 data 19eb, 20, 0c, 1a
20060 data 1a07, 20, 0c, 1a
20070 data 1887, 4c, 6c, 18
20080 data 1890, 4c, 6c, 18
20090 data 17b1, 20, 52, 19
20100 data 17f1, 20, 52, 19
20110 data 1884, 20, 52, 19
20120 data 1948, 20, 52, 19
20130 data 1692, 20, 06, 14
20140 data 16c6, 20, 06, 14
20150 data 1712, 20, 06, 14
20160 data 1747, 20, 06, 14
```

Listing 1. »Graphik-80« (Fortsetzung)



```
20170 data 16e5, 20, 46, 17
20180 data 16fa, 20, 46, 17
20190 data 1740, 4c, b5, 16
20200 data 1869, 4c, 1d, 1a
20210 data 1672, 20, 32, 9e
20220 data 1781, 20, 32, 9e
20230 data 1893, 20, 32, 9e
20240 data 1853, ea, ea, ea
20250 data ende
50000 :
60000 zz$="graphik-80": un=8
60010 open 15,un,15,"s0:"+zz$
60020 gosub 60100
60030 save zz$,un
60040 gosub 60100
60050 verify zz$,un
60060 close 15
60070 end
60100 input#15, s1, s$, s2, s3
60110 if s1=1 then print s2; s$
60120 if s1<20 then return
60130 print s1 ", " s$ "," s2; "," s3
60140 close 15

ready.
```

Listing 1. »Graphik-80« (Schluß)

```
10 fast
20 bank15
30 bload"graphik-80.m"
40 :
50 sys dec("1303")
60 :
90 graphic 6,1
100 circle,320,100,250,99,70,260
110 box, 145,100, 115,110, 160, 1
112 box, 145,100, 115,110,  70, 1
120 circle ,110,45,40,15,1
130 circle,500,40,37,17,,,,45
140 circle,350,100, 160,60, 145,220
150 draw 1, 300,50 to 300,110 to 400,120
 to 300,50
160 circle0,320,100,250,99,70,260
170 circle,320,100,250,99,70,260
182 paint ,320,90
184 circle, 130,85, 200,80, 330,10
186 circle, 509,85, 200,80, 330,10
188 circle, 110,0, 100,55, 158,202
189 circle, 490,0,  97,50, 155,190
190 get q$: if q$<>"" then 200
192 circle1, 340,196, 160,63, 325,40
193 for i=1 to 500: next
194 circle0, 340,196, 160,63, 325,40
195 for i=1 to 500: next
196 goto190
200 graphic 5
220 end

ready.
```

Listing 2. Eine kleine Demonstration der Fähigkeiten von »Graphik-80«.

```
1 rem "                              Setpoint-128.a
2 :
3 rem " zeichnet, l"scht oder testet einen Punkt auf dem 8563-HIRES-Bildschirm
4 :
5 rem "                                (c) by KRW
6 :
7 rem "                                  1985
8 :
9 rem "                               version 1.03
10 :
11 :
20 n$="setpoint-128.3.m"
30 open 15,8,15,"s0:"+n$: gosub 60100
40 open 2,8,2,n$+",p,w": gosub 60100
70 :
80 sys 9*4096
90 ;
100 .opt o2
110 ;
120 ; "*** Varible ***"
130 ;
140 x1l    = $1131            ; "Koordinaten
150 x1h    = $1132            ; "vgl. c-128-Anleitung  S. H-22 !
160 y1     = $1133
165 yh     = $1134
170 zero   = $fd              ; "2-Byte-Zeiger  (fd/fe)
180 zux    = $fc              ; "Arbeitsspeicherstelle"
190 vdc0   = $d600            ; "Adressen der beiden VDC-Register
200 vdc1   = $d601
202 colsel = $83              ; "-ber COLOR-Befehl gew▪hlte Zeichenfarbe
204 ramfig = $9e              ; "Zwischenspeicher f-r RAM-Konfiguration
210 ;
211 xlim      = 639           ; "Bereichsgrenzen
212 ylim      = 199
215 ;
220 ; "*** ROM- Unterprogramme ***"
230 ;
240 setreg   =  $cdcc      ; vdc setzen x=registernummer a=inhalt des registers
250 fechreg  =  $cdd8      ; wert reg. 31 des vdc in akku
270 ;
300 *= $1400
310 ;
900 ↑         jmp setpnt    ; "setzt (colsel<>0) oder l"scht (colsel=0) Punkt
```

Listing 3. Assemblerlisting »setpoint«. Nur zur Dokumentation, nicht eingeben

Ein Byte beim VDC beschreibt acht nebeneinanderliegende Pixels, während ein Byte beim VIC acht untereinanderliegende Pixels kontrolliert. Anders ausgedrückt: Das Pixel mit den Koordinaten 7,0 wird von Bit 0 (!) der ersten Bildschirmspeicherstelle repräsentiert, das Pixel 0,7 von der (7x80=) 560. Speicherstelle. Daraus ergibt sich die folgende Formel zur Berechnung der Bildschirmspeicheradresse für ein gegebenes Koordinatenpaar X,Y:

ADR= Y*80 + INT (X/8)

Zur Berechnung des Bitwertes innerhalb dieser Speicherstelle erhält man (genauso wie beim VIC):

BIT =

2↑ (7 – (»Low-Byte von x« AND 7))

Eine Routine »Punkt setzen« könnte man dann formulieren als:

ADR = (»Inhalt von ADR«) OR BIT

Die entsprechend formulierte Formel für »Punkt löschen» lautet:

ADR = (»Inhalt von ADR«) AND (NOT BIT)

Und schließlich wird noch eine Routine »Punkt testen« für den PAINT-Befehl benötigt:



ERGIBT = (»Inhalt von ADR«) AND BIT

»ERGIBT« wird Null, wenn der Punkt nicht gesetzt ist, und wird ungleich, wenn der Punkt gesetzt ist. Mit diesen Formeln ist ein Programm zum Setzen, Löschen oder Testen eines Bildpunktes auf dem VDC-HiRes-Bildschirm vollständig beschrieben. Das als Listing 3 abgedruckte Maschinenprogramm »setpoint« ist die für das hier beschriebene Grafik-Paket verwendete praktische Formulierung dieses Programms. Dabei wird das Produkt »y*80« nicht berechnet, sondern aus einer Tabelle geladen. Das ist die wahrscheinlich schnellste Möglichkeit, die richtige Bildschirm-RAM-Adresse zu errechnen. Kurze Rechenzeiten aber sind nötig, wenn man bedenkt, wie häufig die Routine aufgerufen werden muß, um eine einzige Kurve auf den Bildschirm zu zeichnen. Alle anderen Teile dieses Programms enthalten keine Besonderheiten. Für den interessierten Leser wurde das Assemblerlisting mit ausführlichen Kommentaren versehen. So dürfte es keine Schwierigkeiten bereiten, eigene Änderungen oder Erweiterungen vorzunehmen.

Komfortable Zeichenbefehle

Bis hierhin wurde recht ausführlich besprochen, wie man den Grafikschirm einschaltet und Punkte setzt. Nun wird sofort der Wunsch wach, Linien, Kreise oder andere Figuren zu zeichnen. Dazu sind schon recht anspruchsvolle Berechnungen nötig. Doch alle Rechenroutinen sind im Basic-ROM des C 128 enthalten. Allen Routinen ist gemeinsam, daß sie im Unterprogramm »Punkt setzen« münden. Die ROM-Routine dafür kann nur Punkte für den VIC-Bildschirm berechnen. Die oben beschriebene Routine bewirkt das gleiche für den VDC-Chip. Der Gedanke liegt nun nahe, in Routinen wie CIRCLE oder DRAW, die Zeiger, die auf das Unterprogramm »Punkt setzen, löschen, testen« zeigen, auf das eigene »Punkt-setzen-Programm« zu »verbiegen«, das den VDC anspricht. Im ROM ist das zwar leider nicht möglich, und die meisten C 128-Benutzer werden ihren neuen 128er nicht gleich mit neuen EPROMS versehen wollen, aber es geht ja auch viel einfacher: Man kopiert einfach die entscheidenden Programmteile ins RAM, paßt die besagten Zeiger an und teilt

```
910 ↑          jmp initram   ; "s. Zeile 920f. !
915 ↑          jmp testpnt   ; "pr┘ft, ob Punkt gesetzt
920 initram    lda $ff00     ; "Init-RAM schaltet RAM-Konfiguration
930 ↑          pha           ; "auf gemeinsamen RAM-Bereich von $0 bis $3fff
940 ↑          lda #$00
950 ↑          sta $ff00
960 ↑          lda #$07
970 ↑          sta $d506     ; "RAM-Konfigurations-Register
980 ↑          pla
990 ↑          sta $ff00
995 ↑          rts
1000 zeiger    lda $ff00     ; "alte RAM-Konfiguration merken
1004 ↑         sta ramfig
1006 ↑         lda #$00      ; "Bank 15 einschalten
1008 ↑         sta $ff00
1010 ↑         sta zero      ; "Arbeitsspeicherstelle l"schen
1020 ↑         sta zero+1
1022 ↑         lda yh
1024 ↑         bne ende
1025 ↑         lda #<ylim
1026 ↑         cmp yl
1027 ↑         bcc ende
1029 ↑         lda #<xlim    ; "Zeile 1022-1033: Bereichsgrenzen pr┘fen
1030 ↑         sbc xl1
1031 ↑         lda #>xlim
1032 ↑         sbc x1h
1033 ↑         bcc ende
1038 ↑         ldy y1        ; "Koordinaten in Bildschirm-RAM-Adresse umrechnen:
1040 ↑         lda faklow,y  ; "1) y-Anteil auswerten
1050 ↑         sta zero      ; "y-Anteil = y-Koordinate * 80
1060 ↑         lda fakhigh,y ; "low-Byte aus Tabelle faklow
1070 ↑         sta zero+1    ; "high-Byte aus Tabelle fakhigh
1140 ;
1150 setpt1    ldx x1h       ; "2) x-Anteil auswerten und zum y-Anteil addieren
1151 ↑         lda x11   ;   ; "low-x = int(x11/8) <Zeile 1151-1155>
1152 ↑         and #%11111000
1153 ↑         lsr
1154 ↑         lsr
1155 ↑         lsr
1157 ↑         clc           ; "x-Anteil = low-x + high-x = zux
1160 ↑         adc divtab,x  ; "high-x aus Tabelle
1170 ↑         sta zux
1250 ↑         clc           ; "Zeilen 1250 bis 1290:
1260 ↑         lda zero      ; "3)  RAM-Adresse = (zero/zero+1) + zux
1270 ↑         adc zux
1275 ↑         sta zero      ; "Zeilen 1300 bis 1330:
1280 ↑         lda zero+1    ; "Berechnetes Byte aus 8563-Bildschirm-RAM holen
1281 ↑         adc #$00
1290 ↑         sta zero+1    ; "Wichtig: erst high-Byte, dann low-Byte !!!
1300 ↑         ldx #$12
1305 ↑         jsr setreg    ; "high-Byte der RAM-Adresse setzen
1310 ↑         ldx #$13
1315 ↑         lda zero
1320 ↑         jsr setreg    ; "low-Byte der RAM-Adresse setzen
1330 ↑         jsr fechreg   ; "Byte holen
1340 ↑         sta zux       ; "und merken
1350 ↑         lda x11       ; "Zeile 1350 bis 1380: Bitzeiger berechen
1360 ↑         and #7
1370 ↑         tax           ; "bit = 2↑((7-x) and 7)
1380 ↑         lda hoch2,x   ; "A enth▪lt bit-Wert
1390 ↑         clc
1400 ↑         rts
1410 ;
1500 ende      sec           ; "Ende, wenn Bereichsgrenzen ┘berschritten
1510 ↑         rts
1540 ;
1700 hoch2     .byt $80, $40, $20, $10
1710 ↑         .byt $08, $04, $02, $01
1720 ;
1730 divtab    .byt $00,$20,$40
1740 ;
3000 faklow    .byt   <0*80,  <1*80,  <2*80,  <3*80,  <4*80,  <5*80
3010 ↑         .byt   <6*80,  <7*80,  <8*80,  <9*80, <10*80, <11*80
3020 ↑         .byt  <12*80, <13*80, <14*80, <15*80, <16*80, <17*80
3030 ↑         .byt  <18*80, <19*80, <20*80, <21*80, <22*80, <23*80
3040 ↑         .byt  <24*80, <25*80, <26*80, <27*80, <28*80, <29*80
3050 ↑         .byt  <30*80, <31*80, <32*80, <33*80, <34*80, <35*80
3060 ↑         .byt  <36*80, <37*80, <38*80, <39*80, <40*80, <41*80
3070 ↑         .byt  <42*80, <43*80, <44*80, <45*80, <46*80, <47*80
3080 ↑         .byt  <48*80, <49*80, <50*80, <51*80, <52*80, <53*80
3090 ↑         .byt  <54*80, <55*80, <56*80, <57*80, <58*80, <59*80
3100 ↑         .byt  <60*80, <61*80, <62*80, <63*80, <64*80, <65*80
3110 ↑         .byt  <66*80, <67*80, <68*80, <69*80, <70*80, <71*80
3120 ↑         .byt  <72*80, <73*80, <74*80, <75*80, <76*80, <77*80
3130 ↑         .byt  <78*80, <79*80, <80*80, <81*80, <82*80, <83*80
3140 ↑         .byt  <84*80, <85*80, <86*80, <87*80, <88*80, <89*80
3150 ↑         .byt  <90*80, <91*80, <92*80, <93*80, <94*80, <95*80
3160 ↑         .byt  <96*80, <97*80, <98*80, <99*80,<100*80,<101*80
3170 ↑         .byt <102*80,<103*80,<104*80,<105*80,<106*80,<107*80
3180 ↑         .byt <108*80,<109*80,<110*80,<111*80,<112*80,<113*80
3190 ↑         .byt <114*80,<115*80,<116*80,<117*80,<118*80,<119*80
3200 ↑         .byt <120*80,<121*80,<122*80,<123*80,<124*80,<125*80
3210 ↑         .byt <126*80,<127*80,<128*80,<129*80,<130*80,<131*80
3220 ↑         .byt <132*80,<133*80,<134*80,<135*80,<136*80,<137*80
3230 ↑         .byt <138*80,<139*80,<140*80,<141*80,<142*80,<143*80
3240 ↑         .byt <144*80,<145*80,<146*80,<147*80,<148*80,<149*80
3250 ↑         .byt <150*80,<151*80,<152*80,<153*80,<154*80,<155*80
3260 ↑         .byt <156*80,<157*80,<158*80,<159*80,<160*80,<161*80
3270 ↑         .byt <162*80,<163*80,<164*80,<165*80,<166*80,<167*80
3280 ↑         .byt <168*80,<169*80,<170*80,<171*80,<172*80,<173*80
```

```
3290 ↑        .byt <174*80,<175*80,<176*80,<177*80,<178*80,<179*80
3300 ↑        .byt <180*80,<181*80,<182*80,<183*80,<184*80,<185*80
3310 ↑        .byt <186*80,<187*80,<188*80,<189*80,<190*80,<191*80
3320 ↑        .byt <192*80,<193*80,<194*80,<195*80,<196*80,<197*80
3330 ↑        .byt <198*80,<199*80,<200*80,<201*80,<202*80,<203*80
3340 ↑        .byt <204*80
3390 ;
3400 fakhigh .byt   >0*80,   >1*80,   >2*80,   >3*80,   >4*80,   >5*80
3410 ↑        .byt   >6*80,   >7*80,   >8*80,   >9*80,  >10*80,  >11*80
3420 ↑        .byt  >12*80,  >13*80,  >14*80,  >15*80,  >16*80,  >17*80
3430 ↑        .byt  >18*80,  >19*80,  >20*80,  >21*80,  >22*80,  >23*80
3440 ↑        .byt  >24*80,  >25*80,  >26*80,  >27*80,  >28*80,  >29*80
3450 ↑        .byt  >30*80,  >31*80,  >32*80,  >33*80,  >34*80,  >35*80
3460 ↑        .byt  >36*80,  >37*80,  >38*80,  >39*80,  >40*80,  >41*80
3470 ↑        .byt  >42*80,  >43*80,  >44*80,  >45*80,  >46*80,  >47*80
3480 ↑        .byt  >48*80,  >49*80,  >50*80,  >51*80,  >52*80,  >53*80
3490 ↑        .byt  >54*80,  >55*80,  >56*80,  >57*80,  >58*80,  >59*80
3500 ↑        .byt  >60*80,  >61*80,  >62*80,  >63*80,  >64*80,  >65*80
3510 ↑        .byt  >66*80,  >67*80,  >68*80,  >69*80,  >70*80,  >71*80
3520 ↑        .byt  >72*80,  >73*80,  >74*80,  >75*80,  >76*80,  >77*80
3530 ↑        .byt  >78*80,  >79*80,  >80*80,  >81*80,  >82*80,  >83*80
3540 ↑        .byt  >84*80,  >85*80,  >86*80,  >87*80,  >88*80,  >89*80
3550 ↑        .byt  >90*80,  >91*80,  >92*80,  >93*80,  >94*80,  >95*80
3560 ↑        .byt  >96*80,  >97*80,  >98*80,  >99*80, >100*80, >101*80
3570 ↑        .byt >102*80,>103*80,>104*80,>105*80,>106*80,>107*80
3580 ↑        .byt >108*80,>109*80,>110*80,>111*80,>112*80,>113*80
3590 ↑        .byt >114*80,>115*80,>116*80,>117*80,>118*80,>119*80
3600 ↑        .byt >120*80,>121*80,>122*80,>123*80,>124*80,>125*80
3610 ↑        .byt >126*80,>127*80,>128*80,>129*80,>130*80,>131*80
3620 ↑        .byt >132*80,>133*80,>134*80,>135*80,>136*80,>137*80
3630 ↑        .byt >138*80,>139*80,>140*80,>141*80,>142*80,>143*80
3640 ↑        .byt >144*80,>145*80,>146*80,>147*80,>148*80,>149*80
3650 ↑        .byt >150*80,>151*80,>152*80,>153*80,>154*80,>155*80
3660 ↑        .byt >156*80,>157*80,>158*80,>159*80,>160*80,>161*80
3670 ↑        .byt >162*80,>163*80,>164*80,>165*80,>166*80,>167*80
3680 ↑        .byt >168*80,>169*80,>170*80,>171*80,>172*80,>173*80
3690 ↑        .byt >174*80,>175*80,>176*80,>177*80,>178*80,>179*80
3700 ↑        .byt >180*80,>181*80,>182*80,>183*80,>184*80,>185*80
3710 ↑        .byt >186*80,>187*80,>188*80,>189*80,>190*80,>191*80
3720 ↑        .byt >192*80,>193*80,>194*80,>195*80,>196*80,>197*80
3730 ↑        .byt >198*80,>199*80,>200*80,>201*80,>202*80,>203*80
3740 ↑        .byt >204*80
3900 ;
4000 setpnt   jsr zeiger    ; "Adresse des zu ▪ndernden Bytes im
4005 ↑        bcs ende1
4010 ↑        ldx colsel    ; "Bildschirm-RAM ausrechnen, bit-Wert nach A
4020 ↑        bne orflag    ; "wenn Farbe = Hintergrundfarbe, dann l"schen
4030 ↑        eor #$ff
4040 ↑        and zux
4050 ↑        .byt $2c
4060 orflag   ora zux       ; "Vordergrundfarbe gew▪hlt, dann setzen
4070 ↑        sta zux       ; "ge▪ndertes Byte zwischenspeichern
4080 ↑        ldx #$12      ; "Register setzen  s.o. !
4090 ↑        lda zero+1
4100 ↑        jsr setreg
4110 ↑        ldx #$13
4120 ↑        lda zero
4130 ↑        jsr setreg
4140 ↑        ldx #$1f
4150 ↑        lda zux       ; "ge▪ndertes Byte in Bildschirm-RAM
4160 ↑        jsr setreg
4170 ende1    lda ramfig    ; "alte RAM-Konfiguration wiederherstellen
4180 ↑        sta $ff00
4190 ↑        rts           ; "fertig.
4200 ;
5000 testpnt jsr zeiger     ; "Koordinaten nach Bildschirm-RAM-Adresse,
5005 ↑        bcs ende1
5010 ↑        and zux       ; "bit-Wert nach A
5020 ↑        beq ende2     ; "Punkt gesetzt ?
5030 ↑        jsr ende1     ; "ja, dann RAM-Konfig. wiederherstellen
5040 ↑        ldx #0
5050 ↑        rts           ; "zur┘ck mit gesetztem Zero-Flag
5060 ende2    jsr ende1     ; "nein, dann RAM-Konfig. wiederherstellen
5070 ↑        ldx #255
5080 ↑        rts           ; "zur┘ck mit Zero-Flag = 0
5090 ;
8000 .end
9000 end
10000 :
60000 zz$="setpoint-128.3.a": un=8
60010 open 15,un,15,"s0:"+zz$
60020 gosub 60100
60030 save zz$,un
60040 gosub 60100
60050 verify zz$,un
60060 close 15
60070 end
60100 input#15, s1, s$, s2, s3
60110 if s1=1 then print s2; s$
60120 if s1<20 then return
60130 print s1 ", " s$ "," s2; "," s3
60140 close 15

ready.
```


Listing 3. Assemblerlisting »setpoint« (Schluß)

zuletzt noch dem Interpreter mit, daß er fortan beim Aufruf der Grafik-Befehle die neuen Routinen »anspringen« soll, wenn der 80-Zeichen-Bildschirm eingeschaltet ist. Die Lösung dieser Aufgabe ist das abgedruckte Basic-Programm (Listing 1). Es stellt alle Programmteile zusammen, die benötigt werden, um die Befehle GRAPHIC, BOX, CIRCLE, DRAW und PAINT auch für den VDC wirksam werden zu lassen. Alle Befehle für den 640x200-Punkte-Bildschirm sind nur im Programm-Modus ausführbar. Das ist deshalb so eingerichtet, weil der 80-Zeichen-Bildschirm im HiRes-Modus zwangsläufig zerstört wird (siehe oben). Die Befehle können im Direkt-Modus also gar nicht vom Bildschirm geholt und interpretiert werden. Man erhielte lediglich »Dreckflecken« auf dem HiRes-Bild.

Probieren Sie einfach einmal im Direkt-Modus die Befehlsfolge »POKE 54784,25: POKE 54785,128« aus! Man kann sehr viel über die Funktionsweise des VDC im Text-Modus lernen. RUN/STOP-RESTORE rückt die Register wieder zurecht.

Der neue GRAPHIC-Befehl

Der GRAPHIC-Befehl wurde um die Funktionen GRAPHIC 6,0 und GRAPHIC 6,1 erweitert. GRAPHIC 6 schaltet den 8563-HiRes-Modus ein. Folgt der »6« eine »1«, so wird der Bildschirm gelöscht, folgt eine »0«, so bleibt der Bildschirm wie er ist. Der Befehl GRAPHIC 6,1 ersetzt auch den in dieser Implementation nicht vorgesehenen Befehl SCNCLR 6. Alle übrigen Informationen zur Implementation dieses Befehls sind in den ebenfalls abgedruckten ausführlich kommentierten Assemblerlistings des Befehls enthalten (Listing 4).

Die »Originalroutine« im ROM befindet sich im Speicherbereich ab $6b5a.

Die Befehle BOX, CIRCLE, DRAW und PAINT

Diese Befehle funktionieren im 8563-Modus genauso wie es im Bedienungshandbuch für die VIC-Grafik beschrieben ist. Einzige Änderung: Die x-Koordinaten dürfen nun im Bereich von 0 bis 639 liegen. Auch die Implementierung dieser Befehle ist denkbar einfach: Nacheinander werden die Programmteile PAINT ($61a8 bis $62b6), BOX ($62b7 bis $6388),

DRAW ($6797 bis $67d6) und CIRCLE ($668e bis $674c) in den RAM-Bereich ab $1672 kopiert (Basic-Programm Zeile 5000 bis 5340). Darunter, in den RAM-Bereich ab $1952, wird der Programmteil »Strecke zeichnen« aus ROM $9b30 bis $9c18 kopiert. Als nächstes werden die neuen Adressen für Unterprogrammaufrufe eingesetzt (WHILE-DO-Schleife). Es folgt schließlich der neue GRAPHIC-Befehl. Im Basic-Programm ist er in Form von DATA-Zeilen abgelegt. Ebenfalls in Form von DATA-Zeilen sind die schon besprochene Routine »setpoint« (RAM $1400 bis $1671) und die Erweiterung der Interpreterschleife im Basic-Text enthalten.

Die Interpreterschleife

Wie teile ich nun dem Interpreter mit, daß er beim Aufruf der Grafikbefehle nun nicht mehr zu den ROM-, sondern zu unseren neuen RAM-Routinen springen soll? Beim Starten eines Programms holt sich der Interpreter – genauso wie beim C 64 – die Adresse, die in den Speicherstellen $308 und $309 abgelegt ist. Er arbeitet dann das Programm ab, das bei dieser Adresse beginnt.

Setzt man in die Speicherstelle $308/09 nun die Adresse des eigenen Programms ein, dann beginnt der Interpreter nach dem Starten eines Programms mit RUN seine Arbeit bei der neuen Adresse. Eben dieses »Verbiegen« des Interpretervektors bewirkt der Befehl SYS DEC ("1303"). Und noch eine wichtige Kleinigkeit enthält die Initialisierungsroutine: die ins RAM kopierten Programmteile enthalten zahlreiche Unterprogrammsprünge (JSR) in ROM-Routinen. Der Prozessor kann diese Sprünge nur richtig ausführen, wenn ihm der RAM-Speicherbereich, in dem unser Programm liegt, und die angesprochenen ROM-Adressen als ein zusammenhängender 64-KByte-Block erscheinen. Für die Zusammenstellung solcher gemeinsamer Bereiche ist die MMU zuständig (siehe Bedienungshandbuch Anhang B). Die Initialisierungsroutine sorgt dafür, daß die MMU im Bereich von $0 bis $1fff immer die RAMs einschaltet, auf der auch unser Programm liegt. Anders ausgedrückt: Auch wenn man über den BANK-Befehl die ROM-Bank 15 ausgewählt hat, nach dem Durchlaufen unserer Initialisierungsroutine liest die CPU die Adressen zwischen $0 uns $1fff immer aus dem RAM-Bereich in Bank 0 aus.

```
1 rem "                    Graphik
2 :
3 rem "          ver▪ndert Ausf┘hrung des graphic-Befehls
4 :
5 rem "                   (c) by KRW
6 :
7 rem "                      1985
8 :
9 rem "                   version 1.00
10 :
11 :
20 n$="graphik.m"
30 open 15,8,15,"s0:"+n$: gosub 60100
40 open 2,8,2,n$+",p,w": gosub 60100
70 :
80 sys 9*4096
90 ;
100 .opt o2
110 ;
220 ; "*** Unterprogramme ***"
230 ;
240 setreg  = $cdcc
250 saverom = $ce0c
270 ;
900 *= $1a3e
910 ;
1000 ↑        cmp #$9e        ; "folgt CLR ?
1010 ↑        bne graph1
1020 ↑        jsr $a022       ; "ja, dann normale Speicheraufteilung zur┘ck
1030 ↑        jsr $0380       ; "chrget
1040 ↑        lda #$00
1050 ↑        sta $d8         ; "Flag f┘r Text-Modus setzen
1060 ↑        rts
1070 graph1   jsr $87f4       ; "Graphik-Modus holen
1080 ↑        cpx #$06        ; "GRAPHIC 6 ?
1090 ↑        beq graphik     ; "ja, dann Sprung
1100 ↑        bcs error       ; "X>6, dann 'illegal quantity'
1110 ↑        txa
1120 ↑        pha             ; "X merken
1140 ↑        lda #$00        ; "bank 15 einschalten
1150 ↑        sta $ff00
1160 ↑        ldx #25         ; "80-Zeichen-Bildschirm auf Text-Modus schalten
1170 ↑        lda #$40
1180 ↑        jsr setreg
1190 ↑        jsr saverom     ; "┘ber Taste 'ASCII/DIN' gew▪hlten Zeichensatz
1200 ↑        lda $d7         ; "ins 8563-CHARRAM kopieren
1210 ↑        pha
1220 ↑        lda #$80        ; "Zeile 1200 bis 1260:
1230 ↑        sta $d7         ; "80-Zeichen-Bildschirm l▀schen
1240 ↑        jsr $c142       ; "kernal-Routine clear
1250 ↑        pla
1260 ↑        sta $d7
1340 ↑        pla             ; "X 'zerst▀rungsfrei' aus Stack laden
1350 ↑        pha
1360 ↑        tax
1370 ↑        jmp $6b6e       ; "weiter mit normalem GRAPHIC-Befehl
1380 error    jmp $7d28       ; "'illegal quantity'
1390 ;
1400 graphik lda #$00         ; "bank 15 einschalten
1430 ↑        sta $ff00
1460 ↑        ldx #25
1470 ↑        lda #$80        ; "8563-HIRES-Mode einschalten
1480 ↑        jsr setreg
1500 ↑        jsr $9e1c       ; "chkcom, byte-Wert nach X
1502 ↑        lda #$00        ; "wieder auf bank 15
1504 ↑        sta $ff00
1510 ↑        cpx #$02        ; "X>1, dann illegal quantity
1520 ↑        bcs error
1525 ↑        cpx #$00        ; "Hires ein ohne L▀schen des Bildschirms ?
1530 ↑        beq ende        ; "dann fertig
1540 ↑        ldy #$40        ; "Zeile 1540 bis 1670: HIRES-Bild l▀schen
1550 ↑        sty $08         ; "$08: 'Blockz▪hler': 40*256 bytes
1560 ↑        ldy #$00
1570 ↑        tya
1580 ↑        ldx #$12        ; "RAM-Zeiger high
1590 ↑        jsr setreg      ; "und
1600 ↑        ldx #$13        ; "RAM-Zeiger low
1610 ↑     0  jsr setreg      ; "auf Anfang = $0000 setzen
1620 ↑        ldx #31         ; "$00 in gew▪hlte RAM-Speicherstelle
1630 empty    jsr setreg      ; "RAM-Zeiger z▪hlt nach 'sta' selbst einen
1640 ↑        dey             ; "Schritt weiter !
1650 ↑        bne empty
1660 ↑        dec $08
1670 ↑        bne empty
1680 ende     rts             ; "fertig
6100 ;
8000 .end
9000 end
60000 zz$="graphik.a": un=8
60010 open 15,un,15,"s0:"+zz$
60020 gosub 60100
60030 save zz$,un
60040 gosub 60100  ,
60050 verify zz$,un
60060 close 15
60070 end
60100 input#15, s1, s$, s2, s3
60110 if s1=1 then print s2; s$
60120 if s1<20 then return
60130 print s1 ", " s$ "," s2; "," s3
60140 close 15

ready.
```

Listing 4. Assemblerlisting »Graphik«. Nur zur Dokumentation, nicht eingegeben

```
1 rem "                     Interpret-128.a
2 :
3 rem "             ver:ndert Interpreter-Schleife
4 :
5 rem "                      (c) by KRW
6 :
7 rem "                         1985
8 :
9 rem "                      version 1.00
10 :
11 :
20 n$="interpret-128.m"
30 open 15,8,15,"s0:"+n$: gosub 60100
40 open 2,8,2,n$+",p,w": gosub 60100
70 :
80 sys 9*4096
90 ;
100 .opt o2
110 ;
220 ; "*** Unterprogramme ***"
230 ;
240 chrget   =  $0380      ; "holt n:chstes Byte aus BASIC-Text
250 execute  =  $4af3      ; "f┘hrt Befehl aus, token in A
260 inter    =  $4af6      ; "holt n:chsten Befehl und f┘hrt ihn aus
270 ;
300 ; "*** Variable ***
310 ;
320 min      = $de    ; "kleinstes token
330 max      = $e9    ; "gr▀tes token
360 ;
900 *= $1300
910 ;
1000 ↑       jmp anfang
1010 ↑       jmp init
1020 ;
2000 init    sei                    ; "pointer f┘r die
2010 ↑       lda #<anfang           ; "Interpreterschleife Prg-Modus
2020 ↑       sta $0308              ; "auf 'anfang' setzen
2030 ↑       lda #>anfang
2040 ↑       sta $0309
2050 ↑       cli
2060 ↑       lda #$00
2070 ↑       sta $ff00              ; "bank 15 einschalten
2080 ↑       lda #$06
2090 ↑       sta $d506              ; "$0-$1fff gemeinsamer RAM-Bereich
2100 ↑       lda #$20
2110 ↑       sta $30
2120 ↑       sta $32
2130 ↑       sta $34
2140 ↑       lda #$00               ; "Variablen-Zeiger entsprechend
2150 ↑       sta $2f                ; "anpassen
2160 ↑       sta $31
2170 ↑       sta $33
2180 ↑       rts
2190 ;
3000 anfang  jsr chrget             ; "Befehl holen
3010 ↑       cmp #min               ; "eines der 'token', die auf
3020 ↑       bcc alt                ; "neue Befehlsadresse f┘hren sollen ?
3030 ↑       cmp #max
3040 ↑       bcs alt
3050 ↑       tax                    ; "ja, dann
3052 ↑       bit $d7           0    ; "auf 40-Zeichen-Modus testen
3054 ↑       bpl vic                ; "40 Z., dann Sprung auf 'normale' Adresse
3060 ↑       lda tablow-min,x       ; "80 Z.: low-Byte
3070 ↑       sta prg+1              ; "                der neuen Befehlsadresse
3080 ↑       lda tabhigh-min,x      ; "      high-Byte
3090 ↑       sta prg+2              ; "                          nach prg+1,+2
3100 route   txa
3110 ↑       jsr chrget             ; "zur┘ck zur Interpreterschleife
3120 prg     jsr $0000              ; "Befehl ausf┘hren
3130 ↑       jmp inter              ; "zur┘ck zur Interpreterschleife
3140 alt     jsr chrget+6           ; "kein Sonderbefehl, 'normale' Schleife
3150 ↑       jmp execute
3900 ;
4000 tablow  .byt $3e, $72, $d7, $81, $93, $8d
4010 ↑       .byt $2b, $56, $58, $e2, $79, $60
4020 ;
5000 tabhigh .byt $1a, $16, $67, $17, $18, $65
5010 ↑       .byt $64, $18, $69, $69, $6a, $69
5020 ;
6000 vic     lda lowtab-min,x       ; "low-Byte
6010 ↑       sta prg+1              ; "           der '40-Z.-Adresse' nach prg
6020 ↑       lda hightab-min,x      ; "high-Byte
6030 ↑       sta prg+2
6040 ↑       jmp route              ; "weiter s.o.
6050 ;
7000 lowtab  .byt $5a, $a8, $d7, $b7, $8e, $8d
7010 ↑       .byt $2b, $97, $55, $e2, $79, $60
7020 ;
7030 hightab .byt $6b, $61, $67, $62, $66, $65
7040 ↑       .byt $64, $67, $69, $69, $6a, $69
7050 ;
8000 .end
9000 end
60000 zz$="interpret-128.a": un=8
60010 open 15,un,15,"s0:"+zz$
60020 gosub 60100
60030 save zz$,un
60040 gosub 60100
60050 verify zz$,un
60060 close 15
60070 end
60100 input#15, s1, s$, s2, s3
60110 if s1=1 then print s2; s$
60120 if s1<20 then return
60130 print s1 "," s$ "," s2; "," s3
60140 close 15

ready.
```


Listing 5. Assemblerlisting »Interpret«. Nur zur Dokumentation, nicht eingeben

Und da taucht nun gleich eine neue Schwierigkeit auf. Wenn die CPU – egal, welche Bank ausgewählt wurde – im Bereich von $0 bis $1fff nur Bytes aus Bank 0 erreicht, dann können auch die auf Bank 1 in eben diesem Bereich abgelegten Variablen nicht mehr gelesen werden. Und umgekehrt, beim Anlegen neuer Variablen, würde unser Programm in Bank 0 überschrieben, weil der Interpreter nicht wissen kann, daß er, obwohl er in Bank 1 schreiben will, in Wirklichkeit doch in Bank 0 schreibt. Deshalb muß der Anfang des Variablenspeicherbereichs auf $2000 gesetzt werden. Das bedeutet den Verlust von 7 KByte Variablenspeicherbereich (mit »?FRE(1)« überprüfbar!); aber anders geht es leider nicht, wenn man ständiges zeitraubendes Umschalten zwischen den Speicherbänken vermeiden will.

Doch nun zur eigentlichen Interpreterschleife. Zunächst wird ein Zeichen aus dem Basic-Text geholt und geprüft, ob es sich um ein Token der in Frage kommenden Grafikbefehle handelt. Ist das nicht der Fall, so fährt das Programm einfach in der alten Interpreterschleife im ROM fort. Ist ein Grafik-Token gefunden, so testet das Programm als nächstes, ob der VIC oder ob der VDC aktiviert ist. Ist der VIC aktiv, so legt sich das Programm die dem Token entsprechende ROM-Adresse zurecht (selbstverändlicher Code hinter dem »JSR« von »JSR $0000«), arbeitet die normale ROM-Routine ab und kehrt in die Interpreterschleife zurück. Ist der VDC aktiv, so werden die den neuen Befehlen entsprechenden RAM-Adressen geholt. Assemblerlisting 5 zeigt die fertige Maschinenroutine für Init und Interpreterprogramm.

Der aufmerksame Leser wird sich wohl schon gefragt haben, warum die Grafik-Routinen gerade in den Bereich ab $1300 »gequetscht« wurden. Ein Grund dafür wurde schon genannt: Die Routinen enthalten sehr viele Sprünge in ROM-Unterprogramme, die es erforderlich machen, daß die gesamten 48 KByte ROM-I/O ab $4000 eingeblendet sind. Will man umständliche und vor allem zeitraubende Bank-Umschaltungen (JSRFAR im Kernel $ff6e) vermeiden, so muß man das Programm in den RAM-Bereich unterhalb von $4000 legen. Es soll jedoch möglich sein, auf beiden Grafik-Bildschirmen gleichzeitig zu arbeiten. Dann verbietet es sich auch noch, den

Bereich von $1c00 bis $4000 zu benutzen, da dort der Farb- und der Bildschirmspeicher für die VIC-HiRes-Grafik angeordnet sind. Bleibt als der einzige freie RAM-Bereich der Bereich von $1300 bis $1bff.

Das »Kochrezept«

Wer bis hier aufmerksam gelesen hat, wird gleichsam als Belohnung eine Menge interessanter Details zur Programmierung seines C 128 erfahren haben. Aber auch ungeduldige Leser sollten spätestens hier einhalten, es folgt die Bedienungsanleitung für das Programm »Graphik-80«:

1) Tippen Sie das Basic-Programm »Graphik 80« (Listing 1) sorgfältig ein! Für diese Arbeit sollten Sie sich Zeit nehmen, damit Ihnen das »DATA-Grab« am Ende des Programms nicht zur Falle wird. Wer einen Assembler zur Verfügung hat – ein C 64-Assembler genügt –, der kann auch die drei Assemblerlistings abtippen, die fertigen Maschinenprogramme speichern und dann die Zeilen 5000 bis 5340 und 10000 bis 14070 durch drei BLOAD-Befehle ersetzen. Die abgedruckten Assemblerlistings (Listings 3 bis 5) können mit dem »Profi Ass«-Assembler direkt übernommen werden, bei anderen Assemblern dürfen in der Regel die Basic-Befehle und die Hochpfeile nicht verwendet werden.

2) Speichern Sie das Basic-Programm auf Diskette. Wenn Sie später Änderungen am Grafik-Paket vornehmen wollen, brauchen Sie es wieder.

3) Starten Sie das Programm mit »RUN«! Das Diskettenlaufwerk (1541 oder 1570/1571) muß beim Start des Programms eingeschaltet und mit einer Diskette versehen sein.

4) Wenn keine Fehlermeldungen aufgetreten sind, und Ihnen auch keine Fehler beim Abschreiben der DATA-Zeilen unterlaufen sind, dann befindet sich jetzt das fertige »Graphik-80«-Paket unter dem Namen »graphik-80.m« auf Diskette und fertig initialisiert im Speicher. Mit Hilfe des kleinen Testprogramms (Listing 2), das noch mit abgedruckt ist, können Sie leicht überprüfen, ob alles richtig gelaufen ist. Von jetzt ab brauchen Sie nur noch als erste (!) Programmzeile »bload "graphik-80.m": sys dec ("1303")« einzugeben und das Grafik-Paket ist aktiviert. Alle Grafik-Befehle beziehen sich jetzt auf den 80-Zeichen-Schirm.

(Th. Rumbach/D. Winkler/ev)

Roulette C 128

Hier präsentieren wir Ihnen eines der ersten Spiele im C 128-Modus. Es zeichnet sich durch alle Setzmöglichkeiten aus, die es beim richtigen Roulette auch gibt.

Das Programm »Roulette 128« (siehe Listing) ist ein Basic 7.0 Programm und kann vom C 128 im C 128-Modus aus mit RUN "ROULETTE 128" von Diskette geladen und gestartet werden. Das Programm lädt keine weiteren Programmteile oder Files nach, und es benötigt keine weitere Peripherie. Es werden nur die 40 Zeichen beziehungsweise die hochauflösende Grafik benutzt, ein spezieller Monitor für 80 Zeichen ist also auch nicht erforderlich.

Das Programm kann neben der Tastatur auch über einen Joystick bedient werden, der, falls vorhanden, in Port 2 zu stecken ist. Nach dem Start wird der FAST-Modus aktiviert, um Variablenfelder einzulesen und um den Grafik-Bildschirm zu erstellen (Bild 1). Der Vorgang dauert etwa 15 Sekunden.

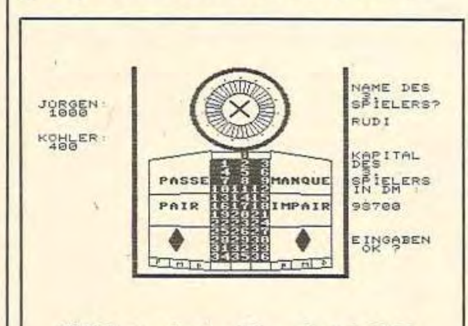


Bild 1. Das hochauflösende Spielfeld

Es können maximal 5 Spieler teilnehmen. Steigt ein Spieler ein, muß er Name und Startkapital eingeben. Nach jedem Spiel können ein oder mehrere Spieler ein- oder aussteigen. Spielt kein Spieler mehr mit, wird das Programm beendet.

Hat ein Spieler sein ganzes Geld verloren, scheidet er automatisch aus. Wenn nun alle Spieler mit den Eingaben fertig sind, wird das Spiel gestartet. Es erscheinen der Name des Spielers, der nun auswählen muß, worauf er wieviel setzt, und die Frage »MENUE ODER TABLEAU« (Tableau = Spieltisch, Bild 1). Jedem Spieler stehen also 2 Wahlmöglichkeiten zur Verfügung, »M« oder »T«. Wird die Taste »M« für Menü gedrückt, wählt der Spieler die Tastatur als Eingabegerät. Dazu werden ihm zuerst alle Sätze angeboten. Nach Wahl eines Satzes kommen auf den Bildschirm die jeweils möglichen Zahlenkombinationen. Mit der »NO-SCROLL«-Taste kann ein eventuelles Herausscrollen aus dem Bildschirm verhindert werden. Der Spieler kann sich dann alle Kombinationen in Ruhe anschauen. Ein weiteres Drücken dieser Taste setzt die Auflistung fort. Jede Zahlenkombination ist mit einer »MENÜ-Zahl« versehen, die anschließend einzugeben ist.

Eine weitere Möglichkeit, dies auszuwählen, besteht darin, über Joystick auf dem Tableau selbst die Auswahl zu treffen. (Bei der Frage »MENUE oder TABLEAU« »T« drücken). Man bewegt dazu einen Chip mit dem Joystick. An der rechten Bildschirmseite werden bei Nichtbewegen des Joysticks der Satz und die Zahlenkombination angezeigt, auf der sich der Chip befindet. Durch Drücken des Feuerknopfes beendet man seine Wahl.

In beiden Fällen (Menü oder Tableau) kommt anschließend die Frage, wieviel man auf diese Zahl beziehungsweise Zahlen setzen will. Die eingegebene Summe muß kleiner gleich dem Kapital des jeweiligen Spielers sein, sonst nimmt das Programm die Eingabe nicht an. Nach Beendigung einer Wahl wird der Spieler gefragt, ob er es bei den bis dahin gewählten Zahlenkombinationen beläßt oder auf noch mehr Zahlen setzen will. Haben sich alle Spieler entschieden, »geht nichts mehr« und die Kugel rollt. Die Zahl, die gewinnt, wird angezeigt, und die Spieler, die auf diese Zahl in irgendeiner Weise gesetzt haben, gewinnen den ihrer Chance entsprechenden Betrag ihres Einsatzes.

Hinweise zum Abtippen

Von Zeile 7000 bis 7499 ist eine Fehlerbehandlungs-Routine eingebaut. Es ist sinnvoll, sie zuerst einzutippen. Durch den Befehl »TRAP 7000« in Zeile 12 wird sie aktiviert.

Grundlagen zu Roulette

Für das Verständnis des Programmes ist es wichtig, wenigstens die Grundbegriffe von Roulette zu kennen.

Aufschlüsselung des Programmes nach Zeilennummern

100 – 146	Daten für Sätze
150 – 172	Daten für Joystick-Ansteuerung
200 – 299	Einlesen der Daten
300 – 999	Zeichen der Grafik
1000 – 1999	Eingabe Name und Kapital des jeweiligen Spielers
2000 – 2999	Eingabe der Spieler, worauf sie setzen. 2030 – 2499 Eingabe mit Joystick 2500 – 2999 Eingabe via Menü
3000 – 3999	Grafische Darstellung der rollenden Kugel, Erzeugung einer Zufallszahl zwischen 0 und 36.
4000 – 4999	Auswertung (Bild 3)
5000 – 5999	Anzeige Kapital, steigt ein Spieler aus? Sprung zum Anfang eines neuen Spieles.
6000 – 6099	Löschen des rechten Grafikbereiches
6100 – 6199	Löschen des linken Grafik-bereiches
6200 – 6999	Eingabe von Name oder Kapital im rechten Grafikbereich; H1$ = kleinstes Zeichen, das eingegeben werden darf; H2$ = größtes Zeichen, das eingegeben werden darf. P2 = Zeile, bei der die Eingabe anfängt B$ = Eingabe
7000 – 7499	Fehlerbehandlungs-Routine
7500 – 7999	Ermittlung des Satzes und der Zahlenkombination aus der Lage des Chips, der mit Joystick gesteuert wurde (Zeile 2030 bis 2499) E$ = Satz, F$ = Zahlenkombination
8000 – 8999	Anzeige von Satz und Zahlenkombination im rechten Grafikbereich
9000 – 9999	Spielende
10000 – xxxx	eigene Erweiterung

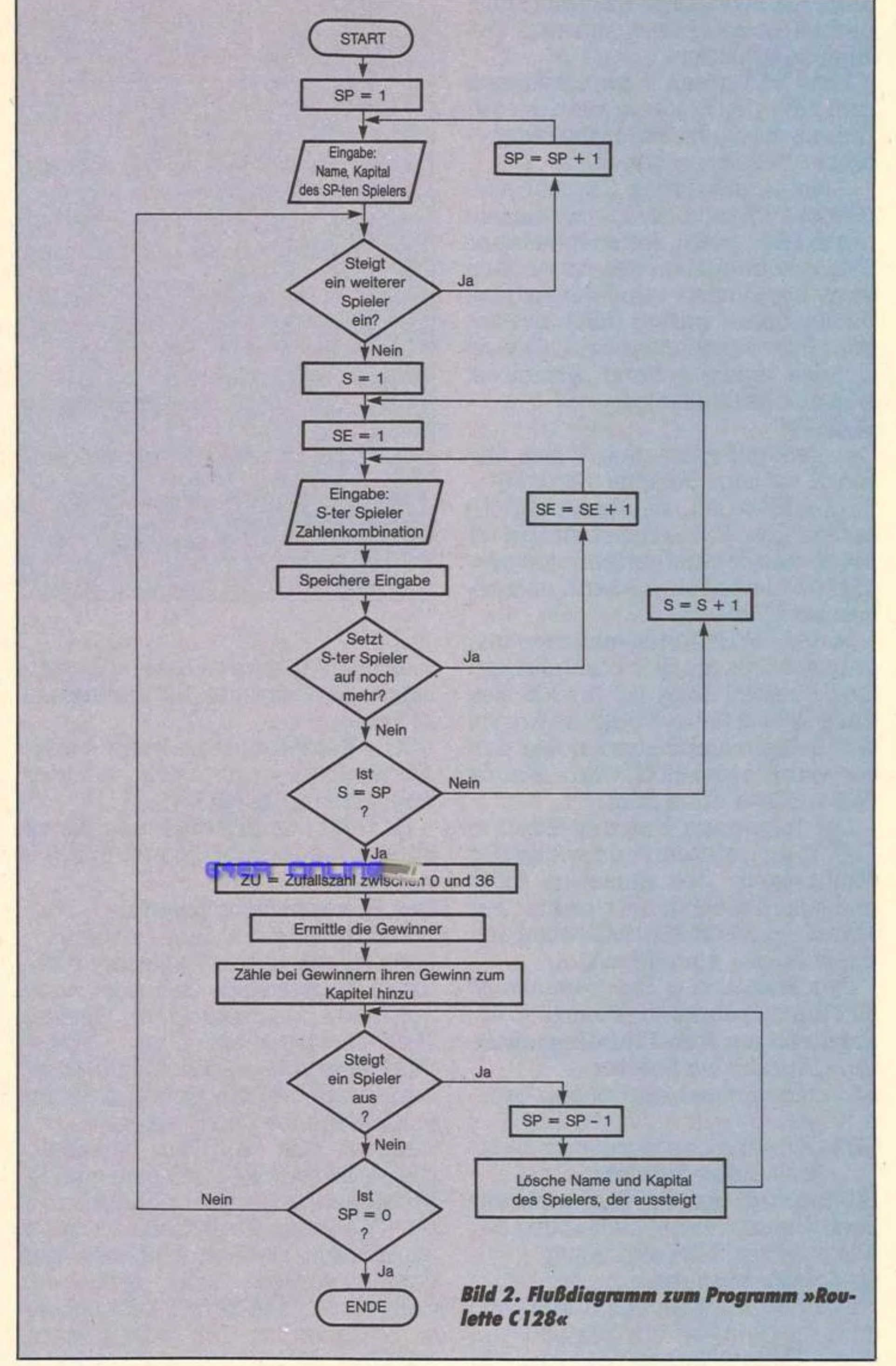


Bild 2. Flußdiagramm zum Programm »Roulette C128«

Als Satz bezeichnet man die 13 Arten, auf Zahlen zu setzen:
Plein (auf eine Zahl)
Cheval (auf 2 Zahlen)
Transversale plein (auf 3 Zahlen)
Carre (4 Zahlen)
Transversale simple (6 Zahlen)
Dutzend (1 bis 12, 13 bis 24 oder 25 bis 36)
Kolonnen (die 3 auf dem Tableau untereinander angeordneten Zahlenreihen, zum Beispiel 1,4,7,10,13,16,19,22,25,28,31,34)
Manque (die Zahlen 1 bis 18)
Passe (die Zahlen 19 bis 36)
Impair (ungerade Zahlen)
Pair (gerade Zahlen)
Rote Zahlen und schwarze Zahlen.

Durch die Anordnung auf dem Tableau gibt es nun zum Beispiel bei »Transversale plein« 14 Möglichkeiten, auf drei Zahlen zu setzen. Im folgenden bezeichne ich dies als mögliche Zahlenkombination bei einem Satz.

Arbeitsweise des Programmes

Ich habe die 13 möglichen Sätze in 10 zusammengefaßt, indem ich zum Beispiel Schwarz und Rot als einen Satz mit zwei möglichen Zahlenkombinationen (zum einen die schwarzen Zahlen, zum anderen die roten) bezeichne. Genauso bei »Manque« und »Passe« und »Impair« und »Pair«.

In Z(X) wurde nun die Anzahl der Zahlenkombinationen des Satzes X abgelegt. Hierzu bekam jeder Satz eine Nummer:
Plein ≙ 1, Cheval ≙ 2, Transversale plein ≙ 3, Carre ≙ 4, Transversale simple ≙ 5, Dutzend ≙ 6, Kolonnen ≙ 7, Manque/Passe ≙ 8, Impair/Pair ≙ 9, Rot/Schwarz ≙ 10.

N$(X):
Beinhaltet den Namen des Satzes X.

S(X):
Beinhaltet den Multiplikator des

Satzes X, mit dem der Einsatz multipliziert werden muß, um den Gewinn zu erhalten.

Hier ist bereits – im Gegensatz zum richtigen Roulette, bei dem der Einsatz noch hinzugezählt wird – dieser bereits enthalten.

Anmerkung: Jeder Satz hat eine gewisse Chance, die mit der Anzahl der Zahlen steigt, auf die mit einer Zahlenkombination gesetzt werden kann. Die Chance bei »Plein« ist zum Beispiel sehr gering (1:36), bei Rot oder Schwarz sehr hoch (1:1). Dieser Chance entsprechend errechnet sich der Multiplikator.

S$(X,Y):
Die Zahlenkombination Y des Satzes X ist zum Beispiel S$ (5,11) = "313233343536", das heißt S$(5,11) enthält die Zahlenkombination 31 bis 36 des Satzes Transversale simple (= 5). Jede Zahl dieser Kombination hat 2 Ziffern.

In den DATA-Zeilen muß man nun je einen Satz als Block betrachten. Zum Beispiel Zeile 105 bis 109. Die erste Zahl in Zeile 105 ist die Anzahl der Zahlenkombinationen des Satzes – in diesem Fall Cheval –, also ist Z(2) = diese erste Zahl.

Die folgenden Daten werden in S$(X,Y) eingelesen. Auf sie folgt der Multiplikator des Einsatzes (S(X)) und als Abschluß der Name des Satzes. Von Zeile 200 bis 240 werden diese Felder eingelesen.

Der Aufbau des Programmes ist im Flußdiagramm zu sehen (Bild 2).

Erklärungen zum Flußdiagramm

SP = Anzahl der Spieler
S = momentan bearbeiteter Spieler
SE = Eingabe des momentan bearbeitenden Spielers.

Da jeder Spieler auf mehrere Zahlen setzen kann, bezeichnet SE, das wievielte Mal er dies tut.

Speichere Eingabe:
Der Spieler wählt mit dem Menü oder mit dem Joystick den Satz und eine Zahlenkombination aus. In A$(S,SE) wird nun zuerst mit zwei Ziffern der Satz gespeichert, dann die Zahlenkombination in der gleichen Form wie in S$(X,Y), gefolgt von dem Zeichen »D« als Ende-Kennzeichen der Zahlenkombination. Aus technischen Gründen folgt dem ein Leerzeichen und schließlich der Einsatz in DM.

Zum Beispiel A$(2,1) = "020912D 50": 2. Spieler, 1. Eingabe auf Cheval, die Zahlen 9/12 mit DM 50,–. In A(X) wird die Anzahl der Eingaben – also SE – vom Spieler X gespeichert.

Ermittle Gewinner
Zähle bei Gewinnern....

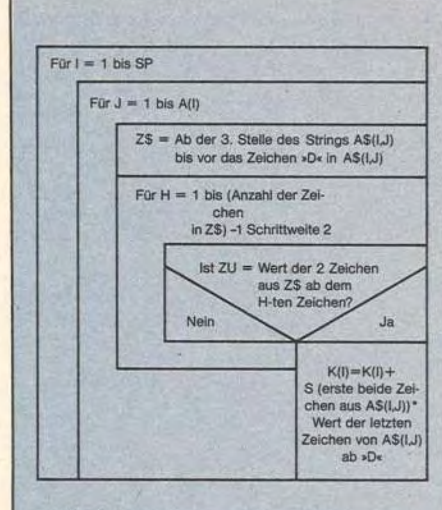


Bild 3. Struktogramm der Anweisungen »ermittle Gewinner« und »zähle bei Gewinnern« aus dem Flußdiagramm.

In Bild 3 ist das zu diesen Anweisungen gehörende Struktogramm zu sehen.

K(X) = Kapital des Spielers X. Diese Auswertung ergibt sich aus der Speicherung in A$(X,Y).

Es sieht komplizierter aus, als es ist, spart jedoch im Computer Speicherplatz.

Zur Auswahl über Joystick

Das Herz des Tableaus stellen die Zahlen 1 bis 36 dar. Sie sind symmetrisch angeordnet, und man kann auf Ihnen die Sätze Plein, Cheval, Transversale plein, Carre, Transversale simple setzen. Diese ergeben zusammen die meisten Zahlenkombinationen des Spieles.

Befindet sich das Chip innerhalb dieses Bereiches, kann man eine X-Koordinate zwischen 0 und 23 und eine Y-Koordinate zwischen 0 und 6 errechnen. Danach läßt sich ein 2-dimensionales Feld aufbauen, das durch diese X- und Y-Koordinate bestimmt ist. Ich wählte dazu X$(X,Y).

Jede Variable dieses Feldes enthält drei Zeichen. Das erste gibt den Satz an und die letzten beiden die wievielte Zahlenkombination gewählt wurde. Somit läßt sich die Zahlenkombination, auf die gesetzt werden soll, eindeutig ermitteln. Für den Satz ist nur ein Zeichen erforderlich, da die möglichen Sätze nur Plein bis Transversale simple umfassen (entsprechen 1 bis 5). Alle Positionen des Chips, die außerhalb dieses Bereiches liegen, es sind insgesamt 13, müssen einzeln abgefragt werden.

(Jürgen Kohler/ah)

```
5 REM     ROULETTE 128
6 REM BY JUERGEN KOEHLER
7 REM     MARTIN-MAY-STR.17
9 REM     6000 FRANKFURT/M 70
11 FAST
12 TRAP 7000
15 SCNCLR:COLOR0,6:GRAPHIC1,1
16 POKE0,PEEK(0) AND63 :REM AMERIK.TAST.
20 DIMZ(10),S$(10,60),S(10),N$(10),K(6),
A$(6,20),A(6),NS$(6),X$(23,6)
30 SP=1:ZU=RND(-TI)
100 DATA37,0,1,2,3,4,5,6,7,8,9,10,11,12,
13,14,15,16,17,18,19,20,21,22,23,24
102 DATA25,26,27,28,29,30,31,32,33,34,35
,36,36,PLEIN
105 DATA60,0001,0002,0003,0102,0104,0203
,0205,0306,0405,0407,0506,0508
106 DATA0609,0708,0710,0809,0811,0912,10
11,1013,1112,1114,1215,1314,1316
107 DATA1415,1417,1518,1617,1619,1718,17
20,1821,1920,1922,2021,2023,2124
108 DATA2223,2225,2324,2326,2427,2526,25
28,2627,2629,2730,2829,2831,2930
109 DATA2932,3033,3132,3134,3233,3235,33
36,3435,3536,18,CHEVAL
110 DATA14,010203,040506,070809,101112,1
31415,161718,192021,222324
111 DATA252627,282930,313233,343536,0001
02,000203,12,TRANSVERSALE PLEIN
115 DATA23,00010203,01020405,02030506,04
050708,05060809,07081011
116 DATA08091112,10111314,11121415,13141
617,14151718,16171920,17182021
117 DATA19202223,20212324,22232526,23242
627,25262829,26272930
118 DATA28293132,29303233,31323435,32333
536,9,CARRE
120 DATA11,010203040506,040506070809,070
809101112,101112131415,131415161718
121 DATA161718192021,192021222324,222324
252627,252627282930,282930313233
122 DATA313233343536,6,TRANSVERSALE SIMP
LE
125 DATA3,010203040506070809101112,13141
5161718192021222324
126 DATA252627282930313233343536,3,DOUZE
130 DATA3,010407101316192225283134,02050
8111417202326293235
131 DATA030609121518212427303336,3,KOLON
NE
135 DATA2,010203040506070809101112131415
161718
136 DATA192021222324252627282930313233334
3536,2,MANQUE/PASSE
140 DATA2,010305070911131517192123252729
313335
141 DATA02040608101214161820222426283032
3436,2,IMPAIR/PAIR
145 DATA2,020406081011131517202224262829
313335
146 DATA01030507091214161819212325273032
3436,2,SCHWARZ/ROT
150 DATA401,201,313,202,314,203,401,301,
102,204,103,206,104,301
152 DATA501,205,402,207,403,208,501,302,
105,209,106,211,107,302
154 DATA502,210,404,212,405,213,502,303,
108,214,109,216,110,303
156 DATA503,215,406,217,407,218,503,304,
111,219,112,221,113,304
158 DATA504,220,408,222,409,223,504,305,
114,224,115,226,116,305
160 DATA505,225,410,227,411,228,505,306,
117,229,118,231,119,306
162 DATA506,230,412,232,413,233,506,307,
120,234,121,236,122,307
164 DATA507,235,414,237,415,238,507,308,
123,239,124,241,125,308
166 DATA508,240,416,242,417,243,508,309,
126,244,127,246,128,309
168 DATA509,245,418,247,419,248,509,310,
129,249,130,251,131,310
170 DATA510,250,420,252,421,253,510,311,
132,254,133,256,134,311
172 DATA511,255,422,257,423,258,511,312,
135,259,136,260,137,312
199 :
200 REM EINLESEN
201 :
205 FORI=1TO10
210 READ Z(I)
215 FORJ=1TOZ(I)
220 READ S$(I,J)
225 NEXTJ
230 READ S(I),N$(I)
235 NEXTI
240 :
245 FORI=0TO23
250 FORJ=0TO6
255 READ X$(I,J)
256 IFLEN(X$(I,J))<>3THENSLOW:PRINT"FEHL
ER IN DATAS AB ZEILE 150!!":END
260 NEXTJ,I
299 :
300 REM GRAFIK
301 :
310 COLOR0,6:COLOR1,1:COLOR4,14
315 GRAPHIC1
330 DATA3,1,3,1,3,1,3,1,3,1,1,3,1,3,1,3,
1,3,3,1,3,1,3,1,3,1,3,1,1,3,1,3,1,3,1,3
331 H=1
335 FORI=11TO22
340 FORJ=17TO21STEP2
345 READA
350 A$=STR$(H):IFH>9THENA$=RIGHT$(A$,2)
355 COLOR1,A
360 CHAR,J,I,A$,1
```



```
365 H=H+1
370 NEXTJ
375 NEXTI
380 COLOR1,1:CHAR,20,10,"0"
382 CHAR,12,13,"PASSE"
384 CHAR,23,13,"MANQUE"
386 CHAR,12,16,"PAIR"
388 CHAR,23,16,"IMPAIR"
395 BOX,137,80,184,87
400 DRAW,137,79TO88,88TO88,188TO137,192T
O184,192TO232,189TO232,88TO184,79
405 DRAW,136,119TO89,120
410 DRAW,136,151TO89,150
415 DRAW,89,180TO137,184TO184,184TO231,1
80
420 DRAW,184,119TO231,120
425 DRAW,184,151TO231,150
430 DATA 104,182,189,120,184,191,136,185
,191,152,185,191,168,185,191,184,185,191
431 DATA 200,184,191,216,182,189
435 FORI=1TO8
440 READ X,Y1,Y2
445 DRAW,X,Y1TOX,Y2
450 NEXTI
455 DRAW,104,163TO109,153TO110,153TO115,
163TO115,164TO110,174TO109,174
460 DRAW,109,174TO104,164
465 PAINT,110,165
466 COLOR1,3
470 DRAW,204,163TO209,153TO210,153TO215,
163TO215,164TO210,174TO209,174
475 DRAW,209,174TO204,164
480 PAINT,210,165
485 COLOR1,1
490 DATA0,0,4,1,0,4,2,0,4,0,1,4,0,2,4,2,
1,4,2,2,4,1,2,4,0,3,4,0,4,4
492 DATA16,3,0,16,4,0,16,5,0,16,6,0,17,3
,0,18,4,0,19,3,0,20,3,0,20,4,0,20,5,0
494 DATA20,6,0
496 DATA32,4,-3,33,4,-3,34,5,-3,34,6,-3,
33,7,-3,32,5,-3,32,6,-3,32,7,-3
500 FORI=1TO29
505 READX,Y,Y1
510 DRAW,94+X,182+Y
515 DRAW,193+X,182+Y+Y1
520 NEXTI
530 FORW=0TO360 STEP(360/15)
531 CIRCLE,160,40,30,,W,W+1
532 NEXTW
533 CIRCLE,160,40,36:CIRCLE,160,40,39
534 PAINT,197,40
535 COLOR1,8
540 CIRCLE,160,40,27
545 CIRCLE,160,40,15
550 COLOR1,16
555 CHAR,19,4,"∨"
556 CHAR,19,5,"∧"
560 COLOR1,8
565 FORW=0 TO 360 STEP(360/37)
570 CIRCLE,160,40,15,,W,W+1
572 X=RDOT(0):Y=RDOT(1)
574 V=(W*π)/180
576 X1=160+27*SIN(V):Y1=40-27*COS(V)
578 DRAW,X,YTOX1,Y1
580 NEXTW
585 COLOR1,10
590 DRAW,75,0TO75,199
592 DRAW,78,0TO78,199
594 PAINT,77,1
595 DRAW,242,0TO242,199
597 DRAW,245,0TO245,199
599 PAINT,244,1
600 DRAW,78,196 TO 242,196
602 PAINT,80,198
605 COLOR1,1
610 CIRCLE,6,6,3:PAINT,6,6
612 SSHAPE A$,0,0,23,20
615 SPRSAV A$,1
620 CIRCLE0,6,6,3:PAINT0,6,6
625 SPRITE1,0,15,0,0,0,0
630 COLOR1,1
635 DRAW,0,0TO1,0TO1,1TO0,1
640 SSHAPE A$,0,0,23,20
645 SPRSAV A$,2
650 DRAW0,0,0TO1,0TO1,1TO0,1
655 SPRITE2,0,1,0,0,0,0
700 POKE0,PEEK(0)OR 64:POKE1,PEEK(1)AND6
3:REM DT.TASTATUR
998 SLOW
999 :
1000 REM EINGABE
1001 :
1002 COLOR1,2
1003 GOSUB6000
1005 CHAR,31,2,"NAME DES":CHAR,31,3,STR$
(SP)+".":CHAR,31,4,"SPIELERS?"
1010 P2=6:H1$="A":H2$="┘":GOSUB6200
1015 NS$(SP)=B$
1020 CHAR,31,10,"KAPITAL":CHAR,31,11,"DE
S":CHAR,31,12,STR$(SP)+"."
1024 CHAR,31,13,"SPIELERS":CHAR,31,14,"I
N DM :"
1025 P2=16:H1$="0":H2$="9":GOSUB6200
1045 K(SP)= VAL(B$)
1050 CHAR,31,20,"EINGABEN":CHAR,31,21,"
OK ?"
1055 GETKEYA$
1060 IFA$="N"THEN1000
1065 IFA$="J"THEN1075
1070 GOTO1055
1075 CHAR,0,SP*4,NS$(SP)+":":CHAR,0,SP*4
+1,STR$(K(SP))
1100 GOSUB6000
1103 IFSP=5 THEN1130
1105 CHAR,31,5,"STEIGT":CHAR,31,6,"NOCH"
:CHAR,31,7,"EIN"
1110 CHAR,31,8,"SPIELER":CHAR,31,9,"EIN
?"
1115 GETKEYA$
1120 IFA$="J"THENSP=SP+1:GOTO1000
1125 IFA$<>"N"THEN1115
1999 :
2000 REM SPIELEINGABE
2001 :
2002 GOSUB6000
2005 FORS=1TOSP
2006 SE=1:GRAPHIC1:GOSUB6000
2008 FORI=1TOSP
2009 CHAR,0,I*4+1,"           ":CHAR,0,I*4
+1,STR$(K(I)):NEXT
2010 CHAR,32,2,NS$(S)
2011 CHAR,32,5,"M",1:CHAR,33,5,"EN┘"
2012 CHAR,33,7,"ODER"
2014 CHAR,32,9,"T",1:CHAR,33,9,"ABLEAU"
2015 CHAR,35,11,"?"
2020 GETKEYA$
2022 IFA$="M"THENGRAPHIC0:GOTO2500
2024 IFA$<>"T"THEN2020
2030 SPRITE1,1
2035 GOSUB6000
2087 AN=0:GOSUB6000
2100 MOVSPR 1,115,229
2110 J=JOY(2)
2111 X=RSPPOS(1,0):Y=RSPPOS(1,1)
2113 IFJ=128THEN2300
2114 IFJ=0ANDAN=1THEN2110
2115 IFJ=0THENGOSUB6000:GOSUB7500:GOSUB8
000:AN=1:GOTO2110
2118 AN=0
2120 IF J=1 THEN BEGIN
2121 :IFX>154ANDX<204ANDY>131ANDY<224THE
NMOVSPR1,+0,-4:GOTO2110
2122 :IFX=120ANDY=185THENMOVSPR1,+14,-46
:GOTO2110
2123 :IFX=115ANDY=215THENMOVSPR1,+5,-30:
GOTO2110
2124 :IF(X=115ORX=130ORX=145)AND(Y=229OR
Y=230ORY=231)THENMOVSPR1,115,215:GOTO211
0
2126 :IFY=232THENMOVSPR1,+1,-9:GOTO2110
2128 :IFX=214ANDY=218THENMOVSPR1,+18,-31
:GOTO2110
2130 :IFX=232ANDY=187THENMOVSPR1,-14,-50
:GOTO2110
2132 :IFY=131THENMOVSPR1,175,127:GOTO211
0
2134 BEND
2140 IFJ=3THEN BEGIN
2142 :IFX>154ANDX<203ANDY>130ANDY<224THE
NMOVSPR1,+8,+0:GOTO2110
2144 :IFX=134ANDY=139THENMOVSPR1,+21,+12
:GOTO2110
2146 :IFX=120ANDY=185THENMOVSPR1,+35,-10
:GOTO2110
2148 :IFX=115ANDY=215THENMOVSPR1,+40,+0:
GOTO2110
2150 :IF(X=115ORX=130)AND(Y=229ORY=230)T
HEN MOVSPR1,+15,+1:GOTO2110
2152 :IFX=145ANDY=231THENMOVSPR1,+17,+1:
GOTO2110
2154 :IF(X=162ORX=178)ANDY=232THENMOVSPR
1,+16,+0:GOTO2110
2156 :IFX=203ANDY>130ANDY<160THENMOVSPR1
,218,137:GOTO2110
2158 :IFX=203ANDY>161ANDY<192THENMOVSPR1
,232,187:GOTO2110
2160 :IFX=203ANDY>195ANDY<223THENMOVSPR1
,214,218:GOTO2110
2165 BEND
2170 IFJ=5THEN BEGIN
2172 :IFX>154ANDX<204ANDY>130ANDY<223THE
NMOVSPR1,+0,+4:GOTO2110
2174 :IFX=175ANDY=127THENMOVSPR1,179,131
:GOTO2110
2176 :IFX=134ANDY=139THENMOVSPR1,120,185
:GOTO2110
2178 :IFX=120ANDY=185THENMOVSPR1,115,215
:GOTO2110
2180 :IFX=115ANDY=215THENMOVSPR1,130,230
:GOTO2110
2182 :IFX=232ANDY=187THENMOVSPR1,214,218
:GOTO2110
2184 :IFX=218ANDY=137THENMOVSPR1,232,187
:GOTO2110
2186 :IFY=223AND(X=163ORX=179ORX=195)THE
NMOVSPR1,-1,+9:GOTO2110
2190 BEND
2200 IF J=7THEN BEGIN
2202 :IFX>155ANDX<204ANDY>130ANDY<224THE
NMOVSPR1,-8,+0:GOTO2110
2204 :IFX=218ANDY=137THENMOVSPR1,203,151
:GOTO2110
2206 :IFX=232ANDY=187THENMOVSPR1,203,175
:GOTO2110
2208 :IFX=214ANDY=218THENMOVSPR1,203,207
:GOTO2110
2210 :IF(X=130ORX=145)AND(Y=230ORY=231)T
HEN MOVSPR1,-15,-1:GOTO2110
2212 :IF(X=178ORX=194)ANDY=232THENMOVSPR
1,-16,+0:GOTO2110
2214 :IFX=155ANDY>130ANDY<160THENMOVSPR1
,134,139:GOTO2110
2216 :IFX=155ANDY>161ANDY<192THENMOVSPR1
,120,185:GOTO2110
2218 :IFX=155ANDY>195ANDY<223THENMOVSPR1
,115,215:GOTO2110
2220 :IFX=162ANDY=232THENMOVSPR1,145,231
:GOTO2110
2222 BEND
2230 GOTO2110
2300 REM BILDEN DES STRINGS
2305 A$(S,SE)=E$+F$+"D"
2315 GOSUB6000
2320 CHAR,31,5,"WIEVIEL":CHAR,31,6,"DM W
OLLEN"
2322 CHAR,31,7,"SIE":CHAR,31,8,"SETZEN ?
"
2325 P2=10:H1$="0":H2$="9":GOSUB6200
2330 IFVAL(B$)>K(S)ORVAL(B$)<=0THEN2315
2335 K(S)=K(S)-VAL(B$)
2340 CHAR,0,S*4+1,"           ":CHAR,0,S*4
+1,STR$(K(S))
2345 A$(S,SE)=A$(S,SE)+" "+B$
2346 A(S)=SE
2347 IFSE=20( 'RK(S)=0THEN2380
2350 GOSUB600(
2355 CHAR,31,1,"WOLLEN":CHAR,31,2,"SIE ,
"
2360 CHAR,31,3,NS$(S):CHAR,31,4,", AUF":
CHAR,31,5,"NOCH MEHR"
2365 CHAR,31,6,"SETZEN ?"
2370 GETKEYA$
2375 IFA$="J"THEN SE=SE+1:GOTO2087
2376 IFA$<>"N"THEN2370
2380 NEXTS
2385 GOTO3000
2500 SCNCLR
2505 SPRITE 1,0
2515 PRINTNS$(S)":":PRINT
2520 FORI=1TO10
2525 PRINTI,N$(I)
2530 NEXTI
2535 INPUT"IHRE WAHL (1,2,3...)";A$
2540 IFVAL(A$)<1ORVAL(A$)>10THEN2535
2545 IFVAL(A$)<10THENA$="0"+A$
2550 SCNCLR
2555 FORI=1TOZ(VAL(A$))
2560 PRINT"¤"I"■ ";
2565 FORJ=1TO(LEN(S$(VAL(A$),I))-1)STEP2
2570 PRINTMID$(S$(VAL(A$),I),J,2)"/";
2575 NEXTJ
2580 PRINT"▮ "
2585 NEXTI
2590 INPUT"IHRE WAHL";B$
2595 IF VAL(B$)<1ORVAL(B$)>Z(VAL(A$))THE
N2590
2600 INPUT"WIEVIEL DM";DM
2605 IF DM>K(S)ORDM<=0THEN2600
2610 K(S)=K(S)-DM
2615 A$(S,SE)=A$+S$(VAL(A$),VAL(B$))+"D"
+STR$(DM)
2617 IFSE=20ORK(S)=0THEN2640
2620 PRINT"WILL SPIELER "S" AUF NOCH MEH
R SETZEN?"
2625 GETKEYC$
2630 IFC$="J"THENSE=SE+1:GOTO2500
2635 A(S)=SE
2640 NEXTS
2999 :
3000 REM ZAHL ZWISCHEN 0 UND 36 ERZEUGEN
3001 :
3002 FORI=1TOSP:CHAR,0,I*4+1,"          "
:CHAR,0,I*4+1,STR$(K(I)):NEXT
3009 SPRITE 1,0
3010 GRAPHIC1
3015 GOSUB6000
3020 COLLISION 2,3300
3050 SPRITE 2,1
3060 MOVSPR2,149,88
3065 R=33:N=R
3100 FORW=(3*π)/2TO(7*π)/2 STEP .3
3105 X=184+R*SIN(W)
3110 Y=88-R*COS(W)
3115 MOVSPR2,X,Y
3120 R=N-.05:N=R
3125 IF R<22ANDRND(1)<.7THEN3500
3130 IFR<19THEN3500
3140 NEXT
3170 GOTO3100
3300 IF BUMP(2)<>2 THEN RETURN
3305 R=R-(RND(1)-.2)
3310 IFN>34 THENN=34
3315 N=R-(RND(1)-.2)
3320 RETURN
3500 SLEEP 2
3505 SPRITE 2,0
3510 SLEEP 1
3515 COLLISION 2
3900 ZU= RND(-TI)
3915 ZU=INT(RND(1)*37)
3919 FORJ=0TO5
3920 FORI=1TO16
3925 COLOR1,I
3930 CHAR,10,5,STR$(ZU):CHAR,26,5,STR$(Z
U)
3935 NEXTI,J
3940 COLOR1,2
3945 CHAR,10,5,STR$(ZU):CHAR,26,5,STR$(Z
U)
3950 SLEEP3
3955 CHAR,10,5,"   ":CHAR,26,5,"   "
3960 SPRITE2,0
3999 :
4000 REM AUSWERTUNG
4005 :
4010 FORI=1TOSP
4015 FORJ=1TOA(I)
4020 :SL=INSTR(A$(I,J),"D")
4025 Z$=MID$(A$(I,J),3,SL-3)
4030 FORH=1 TO(LEN(Z$)-1)STEP2
4035 IFZU=VAL(MID$(Z$,H,2))THEN4060
4040 NEXTH
4045 NEXTJ
4050 NEXTI
4055 GOTO5000
4060 G=VAL(MID$(A$(I,J),1,2))
4065 DM=VAL(MID$(A$(I,J),SL+2,(LEN(A$(I,
J))-SL)+1))
4070 K(I)=K(I)+S(G)*DM
```

Listing zum Programm »Roulette C128«

```
4075 GOTO4045
4999 :
5000 REM ANZEIGE KAPITAL
5005 :
5010 GOSUB6100
5015 FORI=1TOSP
5020 CHAR,0,I*4,NS$(I)+":"
5025 CHAR,0,I*4+1,STR$(K(I))
5030 NEXTI
5035 GOSUB6000
5036 GOSUB10000 :REM EIGENE ERWEITERUNG
5040 CHAR,31,5,"STEIGT":CHAR,31,6,"EIN"
5045 CHAR,31,7,"SPIELER":CHAR,31,8,"AUS
?"
5046 PV=0
5047 FORK=1TOSP
5048 IFK(K)=0THENA$="J":PV=1:GOTO5055
5049 NEXTK
5050 GETKEYA$
5055 IFA$="J"THEN5070
5060 IFA$<>"N"THEN5050
5065 GOTO1100
5070 GOSUB6000
5075 CHAR,31,3,"SEIN":CHAR,31,4,"NAME ?"
5076 IFPV=1THENCHAR,31,6,NS$(K):B$=NS$(K
):GOTO5115
5080 P2=6:H1$="A":H2$="┘":GOSUB6200
5115 S=1
5120 IFB$=NS$(S)THEN5145
5125 IFS<SPTHENS=S+1:GOTO5120
5130 CHAR,31,15,"DIESEN":CHAR,31,16,"SPI
ELER":CHAR,31,17,"KENNE ICH"
5135 CHAR,31,18,"NICHT!"
5140 SLEEP5:GOTO5035
5145 FORI=STOSP
5150 NS$(I)=NS$(I+1)
5155 K(I)  =K(I+1)
5160 NEXTI
5165 SP=SP-1:IFSP=0THENGOTO9000
5170 GOTO5000
5999 :
6000 REM RECHTEN BEREICH DER GRAPHIC
       LOESCHEN
6005 :
6010 FORI=0TO24
6015 CHAR,31,I,"         "
6020 NEXTI
6025 RETURN
6030 :
6100 REM LINKEN BEREICH DER GRAPHIC
       LOESCHEN
6105 :
6110 FORI=0TO24
6115 CHAR,0,I,"         "
6120 NEXT
6125 RETURN
6199 :
6200 REM EINGABE STRING
6201 :
6205 P1=31:B$=""
6210 GETKEYA$
6215 IF((A$>=H1$ANDA$<=H2$)ORA$=" ")ANDP
1<=38THEN BEGIN
6220 CHAR,P1,P2,A$:P1=P1+1:B$=B$+A$:BEND
6225 IFA$=CHR$(13) THENRETURN
6230 IFA$=CHR$(20) AND P1>31THEN BEGIN
6235 P1=P1-1:CHAR,P1,P2," ":B$=LEFT$(B$,
LEN(B$)-1 ):BEND
6240 GOTO 6210
6999 :
7000 REM FEHLERBEHANDLUNG
7001 :
7010 SLOW:GRAPHIC0
7015 COLOR0,12:COLOR5,14
7020 PRINT:PRINTERR$(ER)" ERROR  IN "EL
7025 FORI=1TO8:SPRITEI,0:NEXTI
7030 HELP
7040 END
7499 :
7500 REM FESTSTELLUNG VON SATZ AUS GRAPH
IC
7501 :
7505 X=RSPPOS(1,0):Y=RSPPOS(1,1)
7510 IFX>154ANDX<204ANDY>130ANDY<224THEN
 BEGIN
7515 :E$="0"+LEFT$(X$((Y-131)/4,(X-155)/
8),1)
7520 :F$=RIGHT$(X$((Y-131)/4,(X-155)/8),
2)
7525 :F$=S$(VAL(E$),VAL(F$)):RETURN
7530 BEND
7540 IFX=175ANDY=127THENE$="01":F$="0":R
ETURN
7545 IFX=134ANDY=139THENE$="08":F$=S$(8,
2):RETURN
7550 IFX=120ANDY=185THENE$="09":F$=S$(9,
2):RETURN
7555 IFX=115ANDY=215THENE$="10":F$=S$(10
,1):RETURN
7560 IFX=115ANDY=229THENE$="06":F$=S$(6,
1):RETURN
7565 IFX=130ANDY=230THENE$="06":F$=S$(6,
2):RETURN
7570 IFX=145ANDY=231THENE$="06":F$=S$(6,
3):RETURN
7575 IFX=162ANDY=232THENE$="07":F$=S$(7,
1):RETURN
7580 IFX=178ANDY=232THENE$="07":F$=S$(7,
2):RETURN
7585 IFX=194ANDY=232THENE$="07":F$=S$(7,
3):RETURN
7590 IFX=214ANDY=218THENE$="10":F$=S$(10
,2):RETURN
7595 IFX=232ANDY=187THENE$="09":F$=S$(9,
1):RETURN
7600 IFX=218ANDY=137THENE$="08":F$=S$(8,
1):RETURN
7610 RETURN
8000 REM ANZEIGE AUS GRAPHIC
8001 :
8005 ZE=5
8010 IFVAL(E$)>7THEN8030
8015 IFLEN(N$(VAL(E$)))>8THENBEGIN: CHAR
,31,2,"TRANSVER."
8020 CHAR,32,3,RIGHT$(N$(VAL(E$)),6):GOT
O8030
8025 BEND
8026 CHAR,32,2,N$(VAL(E$))
8030 FORI=1TOLEN(F$) STEP 6
8035 CHAR,32,ZE,MID$(F$,I,2)+" "+MID$(F$
,I+2,2)+" "+MID$(F$,I+4,2)
8037 ZE=ZE+1
8040 NEXTI
8045 RETURN
8999 :
9000 REM SPIELENDE
9001 :
9010 GOSUB6000:GOSUB6100
9020 CHAR,0,2,"WIR":CHAR,0,3,"DANKEN":CH
AR,0,4,"IHNEN"
9025 CHAR,0,5,"FUER":CHAR,0,6,"IHREN":CH
AR,0,7,"BESUCH."
9030 CHAR,0,9,"AUF":CHAR,0,10,"WIEDERSEH
EN"
9040 SLEEP10:GRAPHIC0
9050 END
9999 :
10000 REM EIGENE ERWEITERUNG
10001 :
10010 RETURN

READY.
```

»Roulette C 128« (Schluß).



CP/M auf dem C 128

Der C128 ist mit dem professionellen 8-Bit-Betriebssystem CP/M ausgestattet. Doch was ist eigentlich dieses CP/M und was kann man damit anfangen?

Mit dem C128 können auch Commodore-Besitzer nun CP/M nutzen; das ist etwas, was bisher im Konzept dieses Herstellers gefehlt hat, läßt man einmal den Versuch außer acht, den C64 mit Hilfe eines Z80-Steckmoduls CP/M-fähig zu machen. Dies scheiterte zum einen am Bildschirmformat (nur 40 Zeichen) und zum anderen an der 1541, die nicht in der Lage war und ist, verschiedene Diskettenformate zu lesen. Dadurch können zwar die von Commodore gelieferten CP/M-Systemdisketten und auch andere CP/M-Disketten im 1541-Format gelesen werden, aber es ist leider kein allzu breites Software-Angebot in diesem (CP/M-unüblichen) Format vorhanden. So ist CP/M zwar auch mit der 1541-Station möglich, aber es gibt nur wenig Software dafür.

Beide Nachteile sind beim C128 in Verbindung mit der 1570/71-Floppy nicht mehr vorhanden. Der C128 ist mit der neuesten CP/M-Version, nämlich mit dem CP/M 3.0 (auch CP/M plus genannt) ausgestattet. Dieser Artikel soll keine komplette Einführung in CP/M sein, er soll vielmehr kurz und vollständig erklären, welches Handwerkszeug der Anwender mit dem Betriebssystem CP/M auf dem C128 in die Hände bekommen hat.

Was ist eigentlich CP/M?

CP/M ist ein Betriebssystem und keine »Computersprache«, wie vielfach irrtümlicherweise angenommen wird. Übersetzt heißt es sinngemäß »Kontrollprogramm für Mikroprozessoren«. Dieses Pro-

gramm kontrolliert nun folgende Funktionen des Computers.
- Erkennen und Ausführen aller Eingaben
- Steuerung von Tastatur, Bildschirm, Drucker, Floppy etc.
- Komplette Verwaltung aller Daten auf der Diskette
- Ausführung von Anwender-Programmen.

Das CP/M-System ist praktisch eine universelle Schnittstelle zwischen Anwenderprogramm und Computerhardware. Unter CP/M laufende Programme greifen daher nie direkt auf die Hardware zu, sondern wickeln alle Ein- und Ausgaben indirekt ab, nämlich durch Aufruf einer speziellen Betriebssystemroutine, dem sogenannten »BDOS CALL«. CP/M wurde ursprünglich für den 8080-Prozessor von Intel entwickelt, heutzutage wird allerdings wegen seiner größeren Leistungsfähigkeit fast ausschließlich Zilogs Z80 verwendet.

Der Aufbau von CP/M

Das eigentliche Betriebssystem besteht aus drei Hauptteilen, die beim »BOOTEN« des CP/M-Systems in den Speicher geladen werden. Dazu kommen eine Reihe sogenannter »transienter« Kommandos, das sind Dienstprogramme, die nur bei Bedarf in den Speicher geladen werden.

Der speicherresidente Teil setzt sich zusammen aus dem BDOS (Basic Disk Operating System), einem Grundsystem zur Diskettenverwaltung; dem BIOS (Basic Input/Output System), einem Anpassungsprogramm an die Hardware des Systems, und dem CCP (Console Command Processor), dem Kommando-Interpreter, der für die Ausführung von Benutzerkommandos verantwortlich ist. Unter CP/M laufende Programme sind ohne Schwierigkeiten an jedes Mikrocomputersystem anzupassen, das mit einem Z80-Prozessor läuft. Die große Verbreitung von CP/M hat dazu geführt, daß eine Vielzahl von Programmen zu allen Anwendungsbereichen zu finden ist, ob Textverarbeitung, Datenbanken oder Compiler, für die meisten Programmiersprachen – unter CP/M ist fast alles zu haben.

Um nun mit CP/M zu arbeiten, muß man das System BOOTen. Darunter wird das Kopieren der Systemspuren von der Diskette in den Arbeitsspeicher verstanden. Dazu muß die CP/M-Systemdiskette im Laufwerk A sein. Nach dem Einschalten oder auch nach einem Reset überprüft nun der Computer, ob sich auf der eingelegten Diskette eine CP/M-Spur befindet. Wenn ja, wird weiterhin geprüft, ob die Systemfiles CPM + und CCP vorhanden sind. Danach werden diese beiden Files in den Speicher geladen und das System steht dem Anwender zur Verfügung. Erst ab diesem Zeitpunkt sind die CP/M-Befehle und die Hilfsprogramme verfügbar.

Das ist für viele Anwender bestimmt ungewohnt, aber wie sich noch herausstellen wird, bietet diese Methode, die Diskettenstation als zentrale Anlaufstelle zu benutzen, fast nur Vorteile. Aus diesem Grund kann CP/M auch bis zu vier Diskettenlaufwerke ansprechen; sie werden mit A bis D bezeichnet. Durch die Systemmeldung erfährt der Benutzer, mit welchem Laufwerk er gerade arbeitet. So bedeutet zum Beispiel »B«, daß das gerade angemeldete Laufwerk die Diskettenstation B ist. Ein weiterer Grund, immer wieder auf die Diskette zuzugreifen, ist die Art und Weise, wie CP/M Kommandos bearbeitet und ausführt. Im Speicher befindet sich nämlich neben BDOS, BIOS und dem CCP nur ein Minimalbefehlssatz, die sogenannten residenten Befehle. Diese Befehle sind die meistgebrauchten, deshalb wollen wir sie einmal näher betrachten:

DIR
Mit diesem Befehl wird das Inhaltsverzeichnis der gerade angesprochenen Diskette angezeigt. Dieser Befehl ist vergleichbar mit den Basic-Befehlen DIRECTORY oder »LOAD"$",8«.

DIRSYS
Entspricht DIR, zeigt aber nur die Systemdateien an.

ERASE
Mit diesem Befehl werden Files auf der Diskette gelöscht; entspricht SCRATCH.

RENAME
Umbenennen eines Filenames; entspricht dem gleichnamigen Basic-Befehl

TYPE
Dieser Befehl zeigt den Inhalt einer Datei an; so können zum Beispiel Text-Files oder Pascal-Files durchgesehen werden, ohne erst großartig ein Textverarbeitungsprogramm einzuladen.

USER
Mit diesem Befehl ist es möglich, die Diskette in unterschiedlichen Benutzerbereichen anzusprechen.

Die zweite Gruppe des CP/M-Befehlssatzes sind die transienten Befehle, das heißt, diese Befehle werden nur von Diskette geladen, wenn sie gebraucht werden. Haben sie dann ihre Aufgabe erfüllt, so wird der von ihnen belegte Speicherplatz wieder frei. Solche Befehle erkennt man an der Dateikennzeichnung »COM«. Diese Befehle können aber auch schon größere Programme mit recht komplexen Aufgaben sein, so zum Beispiel der Befehl PIP. Mit ihm ist es möglich, einzelne Dateien oder ganze Disketten zu kopieren oder Dateien auf den Drucker auszugeben. Dazu später mehr.

Weiterhin ist es möglich, den Befehlen sogenannte Optionen mit auf den Weg zu geben; diese ermöglichen dann eine weitere Vielzahl von Operationen, beispielsweise beim SHOW-Kommando die Anzeige von Einzelheiten über Dateien, ob sie schreibgeschützt sind, wann sie erstellt wurden etc.

Im nun folgenden Teil werden die transienten Befehle des CP/M 3.0 etwas näher erläutert, so daß man sich leicht einen Überblick über die Leistungsfähigkeit des Befehlssatzes verschaffen kann.

DATE und **INITDIR**
Mit diesen zwei Befehlen kann man Protokoll führen, wann auf eine Diskette oder eine Datei das letzte Mal zugegriffen worden ist. Weiterhin läßt sich beim C 128 die interne Uhr damit umstellen.

DEVICE
Mit diesem Hilfsprogramm lassen sich die Ein- und Ausgabekanäle, das Bildschirmformat oder die Baudrate ändern und anzeigen.

DUMP
Der Inhalt eines Files wird in ASCII- oder in Hexadezimal-Darstellung angezeigt.

ED
Dieses Programm erlaubt das Erstellen und/oder Ändern von Dateien. Gerade in Verbindung mit SUBMIT wird der Befehl sehr häufig gebraucht.

FORMAT
Wie der Name schon sagt, werden damit Disketten formatiert. Beim C 128 bietet FORMAT mehrere mögliche Formate an, die voll-



ständig menügesteuert angewählt werden können.

GET und PUT
Mit GET werden alle Befehle, die sonst von der Tastatur kommen, aus einer Diskettendatei genommen. Bei PUT wird jegliche Ausgabe (Drucker, Bildschirm) in eine Diskettendatei geleitet.

PATCH
Damit ist es möglich, eventuelle Fehler in Programmen zu beheben oder selbst neue Befehle zu installieren. Beides setzt jedoch eine genaue Kenntnis von CP/M voraus.

PIP
Eines der wichtigsten Dienstprogramme überhaupt. Hauptaufgabe ist das Kopieren von Files. Im letzten Abschnitt wird der genaue Umgang mit diesem Programm anhand eines Beispieles erklärt.

SUBMIT
Oft tauchen Befehlsfolgen auf, die sich immer wieder gleichen, zum Beispiel das Laden mehrerer Files hintereinander oder ähnliches. Die Befehlsfolge wird in einer Datei abgelegt, die die Dateibezeichnung »SUB« trägt. Wird jetzt SUBMIT aufgerufen, so wird die entsprechende SUB-Datei automatisch ausgeführt. Erstellt werden die Dateien zum Beispiel mit ED oder einer Textverarbeitung.

SID
Ein symbolischer Debugger zum Auffinden von Fehlern in Programmabläufen.

SHOW
Wem der DIR-Befehl zuwenig Informationen über die Diskette gibt, der bekommt eine ganze Menge Zusatzinformationen wie freier Platz etc.

SET
Mit diesem Befehl ist es unter anderem möglich, Files als schreibgeschützt zu kennzeichnen, Datumseinträge vorzunehmen oder ein Paßwort zu definieren.

SETDEF
Suchkriterien und Laufwerksbezeichnungen können nach Vorgaben des Benutzers einander zugeordnet werden.

SAVE
Die Speicherung von Teilen des Speichers wird damit durchgeführt; es lassen sich zum Beispiel Maschinenprogramme auf diese Art abspeichern.

LINK
Damit werden Programmteile zu einem lauffähigen Ganzen verbunden.

MAC, RMAC, HEXCOM, XREF
Diese Dienstprogramme sind für den erfahrenen Programmierer von Nutzen, mit ihnen wird die Programmierung des Z80 unter CP/M ermöglicht.

Diese Kurzbeschreibung erhebt keinen Anspruch auf Vollständigkeit, sie soll jedoch einen kurzen Überblick über die vielfältigen Möglichkeiten des CP/M 3.0 geben. Die detaillierte Beschreibung zu allen Befehlen geben zum Teil die Handbücher, zum Teil die zu CP/M erschienene Literatur.

Kopieren leichtgemacht

Für den Anwender stellt sich oft das Problem, eigene bootfähige Disketten herzustellen. Der Computer verlangt nach dem Einschalten die Dateien »CPM+.SYS« und »CCP.COM«. Sind beide Files auf der Diskette vorhanden, so wird CP/M plus geladen.

Um beide Files zu kopieren, benutzt man PIP.COM. Dieses Programm eignet sich hervorragend, um eine ganze Diskette oder einzelne Files zu kopieren. In den nun folgenden Schritten wird zuerst ein kompletter Backup der Systemdiskette und dann das Anlegen einer bootfähigen Diskette (für eigene Anwendungen) beschrieben.

Als erstes ist festzustellen, daß die beschriebenen Arbeitsschritte sowohl für die 1541 als auch für die 1570/71 ihre Gültigkeit haben, der einzige Unterschied liegt in den unterschiedlichen Ausführungszeiten, die in der untenstehenden Tabelle zu finden sind. Eine weitere Voraussetzung ist das Vorhandensein einer im jeweiligen Format fertig formatierten leeren Diskette. Sollte das noch nicht geschehen sein, so erstellt man sich eine mit Hilfe des COM-Files FORMAT.

Daß eine Mehrzahl der Anwender nur über ein Laufwerk verfügt, spielt im folgenden keine Rolle, da das CP/M des C 128 in der Lage ist, ein »virtuelles« Laufwerk E anzusprechen, was praktisch einen Diskettenwechsel darstellt.

Nachdem PIP.COM geladen wurde, wird nun »e:=a:*.*« eingegeben. Das »e« bedeutet hier das Ersatzlaufwerk, sprich Diskettenwechsel. Sobald sich PIP nach dem Kopieren wieder mit »*« meldet, ist eine Kopie der Systemdiskette erstellt.

Soll aber nur eine bootfähige Diskette erstellt werden, so genügt es, wenn sich auf der Diskette die Files CPM+.SYS und CCP.COM befinden. Um diese zu kopieren, wird wieder PIP geladen. Mit »e:=a:cpm+.sys« und »e:=a:ccp.com« werden die beiden Files auf eine leere, aber schon formatierte Diskette kopiert. Diese Diskette wird jetzt vom Computer als Boot-Diskette anerkannt und CP/M wird nach dem Einschalten und bei eingelegter Diskette geBOOTet.

Im übrigen ist das obengenannte Verfahren nicht nur auf diese beiden Files anwendbar, sondern so lassen sich beliebige Files kopieren, so daß der Anwender seine Disketten beliebig gestalten kann.

(Udo Reetz/ev)

	1571	1541
Booten	20 sec	2 min
Formatieren	50 sec (DS)	90 sec
Systemdisk kopieren	2 min 30	50 min

Ein Zeitvergleich zwischen 1541 und 1571-Floppy unter CP/M



CP/M macht's möglich: Die Textverarbeitung der Profis auf dem C128.

Test: WordStar

Mit WordStar steht dem C128-Besitzer ein Textverarbeitungsprogramm aus dem Bereich der echten Personal-Computer zur Verfügung. Um es vorweg zu sagen: WordStar ist ganz sicher nicht die schnellste Textverarbeitung, die für einen Computer wie den C128 denkbar ist, aber die Leistungsfähigkeit dieses professionellen Programms macht das Geschwindigkeitsmanko mehr als wieder wett.

Doch beginnen wir ganz von vorn: WordStar für den C128 wird auf zwei Disketten geliefert.

Eine Diskette (Nummer 2) enthält das uninstallierte WordStar, eine Art Rohversion, die mittels eines ebenfalls mitgelieferten speziellen »Install«-Programms an den jeweiligen Computer (hier den C128) angepaßt werden kann. Damit braucht man sich bei einem späteren Wechsel des Computers sein Textverarbeitungsprogramm nicht ein zweites Mal zu kaufen: Mit Hilfe des Install-Programms und des sehr ausführlichen Handbuches kann man WordStar an jeden CP/M-fähigen Computer anpassen. Sollte man gar bis in die Höhen der IBM-kompatiblen 16-Bit-Computer umsteigen, dann kann man zwar das für den Z80-Prozessor geschriebene WordStar nicht mehr verwenden, aber man kann mit dem 16-Bit-WordStar zumindest alle »alten« Texte weiterbearbeiten. Diese erste Diskette sollte man daher sehr gut verwahren.

Centronics-Schnittstelle eingebaut

Sollten Sie jetzt etwas Angst vor der vielleicht doch nicht so ganz einfachen Installierung von WordStar auf Ihrem C128 bekommen haben, so ist das absolut unnötig: Die zweite Diskette enthält WordStar schon fix und fertig installiert und an den C128 angepaßt – und das gleich in zwei Versionen: Das File WSCBM.COM ist eine zu Commodore-Druckern kompatible WordStar-Version. Falls Sie also einen Original-Commodore-Drucker der MPS-Reihe besitzen, oder aber Ihren Drucker mittels Centronics-Interface am seriellen Bus betreiben, dann ist diese Version die richtige für Sie.

Die zweite WordStar-Version, WSPAR.COM, unterscheidet sich von der ersten dadurch, daß hier softwaremäßig eine Centronics-Schnittstelle über den User-Port realisiert wurde. Zum Anschluß eines beliebigen Druckers mit Centronics-Schnittstelle braucht man dann nur noch ein entsprechendes User-Port-Kabel anstatt eines teuren Interface.

Bevor man nun mit einer der beiden WordStar-Versionen arbeiten kann, ist allerdings noch etwas Vorarbeit zu leisten: Die WordStar-Disketten sind im 1541-Floppy-Format aufgezeichnet und können daher gleichermaßen mit den Commodore-Laufwerken 1541, 1570 und 1571 gelesen werden. Um jedoch die Vorteile der 1570/1571-Stationen (schnellerer Zugriff, höhere Speicherkapazität) ausnutzen zu können, müssen die Disketten zunächst auf das jeweilige Diskettenformat umkopiert werden.

Sicherheitskopie – ja, bitte!

Umkopiert? Das mag manchem Besitzer anderer professioneller Textverarbeitungssysteme für den C64/C128 etwas merkwürdig vorkommen, aber es hat schon seine Richtigkeit: WordStar ist ebenso wie praktisch alle professionellen CP/M-Programme nicht mit irgendwelchen profilneurotischen Kopierschutz-Pfuschereien ausgestattet, sondern gestattet dem Käufer das Anfertigen von Sicherheitskopien für den persönlichen Gebrauch. Das Handbuch meint zu diesem Thema in der Einleitung: »Arbeiten Sie niemals mit Ihrer Original-Diskette, damit Sie sich bei Beschädigung Ihrer Diskette eine neue Arbeitsdiskette erstellen können«.

Das Umkopieren der Diskette auf das jeweilige Floppy-Format (und damit die Herstellung einer Sicherheitskopie) wird auf mehreren Seiten sehr ausführlich Schritt für Schritt beschrieben.

Bild 1. Das Hauptmenü zeigt neben dem Inhaltsverzeichnis der Diskette auch alle Befehle, in deutsch erklärt an.

Hat man nun eine lauffähige Kopie der WordStar-Diskette erzeugt, kann man leider noch nicht sofort loslegen: Vor dem Start von WordStar muß noch das SETUP-Programm gestartet werden, da das Commodore CP/M unverständlicherweise keine deutschen Umlaute unterstützt. Also heißt es in den sauren Apfel beißen und erst einmal SETUP laufen lassen, ehe man – endlich – WordStar selbst starten kann.

Jetzt sollte allerdings nichts mehr im Wege stehen, sich in die Arbeit mit Wordstar zu stürzen. An dieser Stelle muß jedoch noch einmal von der Verwendung einer 1541 als

Floppy-Laufwerk abgeraten werden: Die 1541 ist einfach viel zu langsam.

So arbeitet es sich mit WordStar

WordStar meldet sich nach dem Laden mit einem übersichtlichen Startmenü (siehe auch Bild 1). Da es die wichtigsten Befehle, das heißt diejenigen, die unentbehrlich sind, um mit der Schreiberei anfangen zu können, klar und eindeutig auflistet, erspart dieses Menü den Griff zum Handbuch. Gerade dem Neuling kommt das zugute, denn der Umgang mit dem Handbuch erfordert einiges an Routine und Zeit. Nebenbei gesagt ist man, wird auf die Schnelle ein besonderer Befehl gesucht, mit der mitgelieferten Referenzkarte vielfach besser als mit dem sehr ausführlichen Handbuch bedient. Die Referenzkarte gibt eine Übersicht über alle zur Verfügung stehenden Kommandos.

So ist im Startmenü aber leicht der Befehl zum Dateieröffnen zu finden: Nur den Buchstaben »D« (wie »Datei«) drücken. Nach Eingabe eines Namens kann man schließlich mit der Abfassung seines Textes beginnen.

Die obere Hälfte des Bildschirms enthält jetzt das sogenannte »Hauptkommando-Menü«, das dem Benutzer ständig die wichtigsten Befehle präsentiert (siehe auch Bild 2). Ein Untermenü von einer Zeile Länge hält alle übrigen WordStar-Funktionen bereit, die problemlos über CTRL-J, CTRL-K, CTRL-O, CTRL-P und CTRL-Q aufzurufen sind. Jeder Einsteiger wird für diese Gedächtnisstütze recht dankbar sein, da er ja wahrscheinlich noch meilenweit davon entfernt ist, alle relevanten Befehle im Kopf zu haben. Denn spätestens nach dem ersten Tippfehler bedarf es der Kenntnis von Cursor- und Löschfunktionen.

Grundsätzlich sind bei WordStar sämtliche Befehle über eine mehrfache Tastenbedienung (»CTRL«-Taste plus Buchstaben oder Buchstabenkombination) auszuführen. Eine Ausnahme bilden hier die Funktionen des Startmenüs, sowie die Steuerungsbefehle des Cursors, da beim C 128 dazu spezielle Cursortasten vorgesehen sind. Anfangs ist es recht ungewohnt, jeweils mehrere Tasten drücken zu müssen, um als Resultat eine einzige Funktion zu erhalten. Da die enorme Vielfalt der erreichbaren Funktionen jedoch ein großes Anwendungsspektrum eröffnen, gewöhnt man sich gerne daran. Das Löschen von Buchstaben und ganzen Wörtern, dessen Control-Code wir dem oben stehenden Menü entnehmen, erfolgt ebenso unproblematisch wie die übrigen aufgelisteten Kommandos. Trotz dieser leicht verständlichen Handhabung sei unbedingt zu empfehlen, alle im Menü vorhandenen Befehle des öfteren durchzuexerzieren. Ohne Übung nämlich kein Meister und ohne Fleiß kein Preis. Hat man sich aber einmal die wichtigsten Funktionen gemerkt, dann geschehen Textkorrekturen sehr schnell und effektiv: Es gibt Funktionen zum zeichen-, wort,- zeilen- oder abschnittsweisen Weitergehen im Text, man kann wahlweise durch den Text »blättern« oder rollen, Zeichen, Worte oder Zeilen löschen. Auch Blockoperationen stehen zur Verfügung: Man kann beliebige Textabschnitte als Blöcke markieren, diese dann mit einfachen Befehlen löschen, kopieren, verschieben, speichern oder als Textbausteine definieren. Automatisches Suchen von Zeichenfolgen im gesamten Text oder auch nur in bestimmmten Abschnitten ist sehr einfach möglich. Auch Suchen mit automatischem Ersetzen der gefundenen Zeichenfolge durch eine andere ist möglich; wahlweise mit oder ohne Rückfrage beim Benutzer. Also frisch an's Werk! Sie werden merken, wie Sie das System mit etwas Übung besser und besser beherrschen, bis Sie schließlich die Editierfunktionen gar nicht mehr bewußt ausführen.

Haben Sie die gängigsten Befehle schon im Kopf? Dann fällt es Ihnen sicher auf, daß das Anzeigenmenü - jetzt wo Sie es nicht mehr so dringend brauchen - eigentlich viel zuviel Platz auf dem Bildschirm einnimmt. Für den Text bleibt lediglich die untere Monitorhälfte (Bild 2), was natürlich auf Kosten der Übersichtlichkeit geht. Dem ist aber problemlos abzuhelfen, vorausgesetzt man weiß wie. Schlagen Sie Ihr Handbuch auf und suchen Sie, denn im Hilfsmenü ist nichts zu finden. Aber früher oder später stößt man auf den Befehl CTRL-J und kann darauf hin mit »h« das Hilfsmenü aufrufen. Hier ist eine Regulierung des Menü-Umfanges von 0 – 3 möglich. Bei Stufe 0 endlich steht Ihnen als Betätigungsfeld der komplette Bildschirm zur Verfügung, lediglich eine einzelne Statuszeile am oberen Bildschirmrand bleibt erhalten.

Bild 2. Während des Bearbeitens eines Textes hat man stets die wichtigsten Kommandos auf einen Blick.

Viel mehr als nur eine elektronische Schreibmaschine

Doch damit sind die Fähigkeiten von WordStar noch lange nicht erschöpft. Auf die beeindruckendsten Funktionen, die WordStar zu WordStar machen, soll im folgenden eingegangen werden.

Ein Charakteristikum von WordStar ist beispielsweise die Möglichkeit, dem einmal eingegebenen Text eine beliebige äußere Form zu geben. Das Eingeben des Briefes, Artikels etc. ist mit den angesprochenen Editierfunktionen völlig problemlos. Die Zeilenschaltung entfällt ebenso wie irgendeine Worttrennung. Ist der Text schließlich »im Kasten«, dann ist es im nachhinein freigestellt, nach Lust und Laune (beziehungsweise dem besten optischen Eindruck gemäß) den Text zu gestalten. In frei gewählter Zeichenbreite – auch

über die 80 Zeichen des Bildschirms hinaus –, links und rechtsbündig als Blocksatz oder als Flattersatz nur linksbündig, kann man den Textblock formatieren. Auch das Zentrieren von Überschriften etc. ist kein Problem. Die Ausführung übernehmen die CTRL-Befehle CTRL-OL, CTRL-OR für die Randbegrenzung. Der Befehl CTRL-OJ, der wie ein Ein/Ausschalter zu bedienen ist, stellt auf Flattersatz um. Endgültig formatiert wird endlich mit CTRL-B. Hierbei ergibt sich ein kleines Problem, nämlich die Worttrennung: Dadurch, daß bei der Texteingabe keinerlei Trennung durchgeführt wurde, konnten zwischen den einzelnen Wörtern – oft sehr unschön - größere Abstände entstehen, wenn rechts- und linksbündig geschrieben wurde. Doch WordStar bietet hier eine Lösung an, die sogenannte »Trennhilfe«, die während des Formatierens zur Anwendung kommt. Sie läßt sich über das CTRL-O-Menü an- und ausschalten. Bei zu langen Wörtern veranlaßt sie eine Unterbrechung des Formatiervorgangs und macht an entsprechender Stelle einen Trennvorschlag. Sind Sie damit einverstanden, daß heißt, ist der Vorschlag orthografisch richtig, setzen Sie den Trennstrich. Ist keine Trennung erwünscht, brauchen Sie nur erneut CTRL-B eingeben. Der Formatierbefehl wirkt immer nur auf einen Absatz, so daß verschiedene Textteile völlig verschieden formatiert werden können. Soll der gesamte Text neu formatiert werden, dann hält man die Tastenkombination CTRL-B etwas länger gedrückt. Die Formatbefehle stellen auch eine nicht zu verachtende Hilfe beim Erstellen von Tabellen, Listen und ähnlichem dar. Wollen Sie beispielsweise eine längere Zahlenkolonne linksbündig in der 10. Spalte auflisten, geben Sie CTRL-OL ein. WordStar fragt nun nach, in welche Spalte Sie Ihre Liste plazieren wollen. Antworten Sie mit »10«, werden die gesamten nachfolgenden Angaben bis zum nächsten RETURN in die zehnte Spalte gerückt. An dieser Stelle wäre auch noch die Tabulatorfunktion zu erwähnen, die während des Schreibens zur Gestaltung von Tabellen etc. eingesetzt werden kann.

Fußnoten – kein Problem

Weiter zu einer Fähigkeit von WordStar, auf die niemand, der schon einmal damit gearbeitet hat, verzichten wollen wird: Gemeint ist das sogenannte »Punkt-Kommando« ».FO«. Punkt-Kommandos steuern die Ausgabe des Textes in einem bestimmten Druckbild. Das ».FO«-Kommando bewirkt, daß Sie eine Fußnote, die sonst mühsam unterhalb des Textblockes eingefügt werden müßte, problemlos im Gesamttext mitschreiben können. ».FO« an den Anfang einer Zeile gesetzt, läßt die restliche Zeile als Fußnote am Blattende erscheinen.

Andere Punktbefehle steuern Seitennumerierung, Hoch- und Tiefstellen von Zeichen und legen die Randbegrenzungen für den Ausdruck fest. Außerdem können Kopfzeilen definiert werden, die zu Anfang jeder neuen Seite immer wieder gedruckt werden.

Eine weitere sehr interessante Anwendung ist die Definition von »Textvariablen«. Das ist wichtig für das Erstellen von Serienbriefen, bei denen nur bestimmte Textteile, in der Regel Namen und Anrede, geändert werden müssen. WordStar erlaubt es sehr einfach, Textvariablen zu verwenden, die erst beim Ausdruck durch entsprechende Daten – zum Beispiel aus einer Adreßdatei – ersetzt werden.

Serienbriefe und Textbausteine

Auf diesem Grundgedanken basiert nun ein mitgeliefertes Programm, nämlich »MailMerge«, das einen wesentlichen Teil der in einem Büro anfallenden Schreibarbeiten erspart. MailMerge erlaubt es, auf einfache Weise auf Adreßdateien oder Textbaustein-Dateien zuzugreifen und erweitert somit die Anwendungsgebiete von WordStar ganz enorm. Der WordStar-Besitzer kann diese Funktionen von Anfang an in Anspruch nehmen: MailMerge befindet sich ganz einfach auf der WordStar-Diskette und ist im Handbuch ausführlich dokumentiert.

Fazit

WordStar ist ganz ohne Zweifel eines der leistungsfähigsten Textverarbeitungs-Programme, die je für einen Home-Computer zu haben waren. Fähigkeiten wie die automatische Verwaltung von Fußnoten oder das Verarbeiten von Textbausteinen heben das Programm über das Niveau so mancher anderen Textverarbeitung hinaus. Das sehr umfangreiche Handbuch im professionellen Ringordner beantwortet alle Fragen, die im Zusammenhang mit der Benutzung oder der Installation von WordStar auftreten können.

Der größte Vorteil von WordStar liegt in der weitgehenden Unabhängigkeit von einem bestimmten Computersystem. Ein- und dasselbe Programm, einmal gekauft, überdauert dank Kopierbarkeit nicht nur beliebig viele Arbeitsdisketten, sondern bleibt auch bei Anschaffung eines neuen Computers noch aktuell – sofern der Computer CP/M-fähig ist. Und selbst beim »Aufstieg« in die Höhen der 16-Bit-Welt der IBM-kompatiblen Computer können die alten Texte und Dateien weiterverwendet werden. Die 199 Mark für dieses Textprogramm sind also auch langfristig gesehen gut angelegt – WordStar kann eine Anschaffung fürs Leben sein.

(Eva-Maria Hierlmeier/tr)

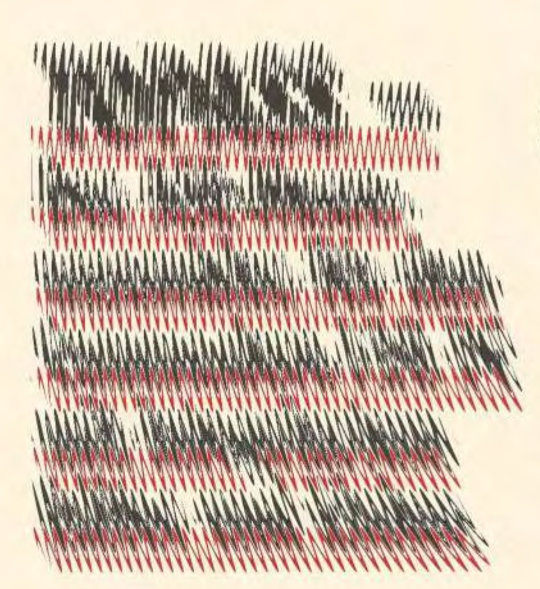
Info: WordStar 3.0 mit MailMerge für den Commodore 128 PC, 199 Mark, Markt & Technik Verlag Aktiengesellschaft, Hans-Pinsel-Str. 2, 8013 Haar bei München

WordStar, eine Textverarbeitung, die keine Wünsche mehr offen läßt?

Daß WordStar aufgrund seiner wirklich umfangreichen Möglichkeiten ein Programm der Spitzenklasse ist, bleibt außer Frage. Lediglich auf dem C 128 kann es nicht voll befriedigen: Vor allem das Rollen durch einen längeren Text, um eine bestimmte Text-Stelle zu suchen, kann aufgrund der sehr langsamen Scroll-Routine entnervend wirken. Der Grund hierfür ist allerdings nicht bei den Programmierern von WordStar, sondern bei Commodore selbst zu suchen. Das BIOS (das ist der Teil des CP/M-Systems, der unter anderem die Bildschirmausgaben steuert) ist, was die Geschwindigkeit anbetrifft, auf dem C 128 alles andere als gut angepaßt. Wer schon einmal WordStar zum Beispiel auf einem Schneider-Computer unter CP/M gesehen hat, wird bestätigen, daß es auch anders geht.

(tr)







64ER ONLINE

Multiplan



Bild 1. Das leere Arbeitsblatt von Multiplan

Multiplan - das ist ein Kalkulationsprogramm, das auf dem Commodore 128 mit dem Betriebssystem CP/M läuft. Wir zeigen Ihnen, wie es sich mit diesem Programm arbeiten läßt.

Tabellenkalkulationen sind Programme, die es dem Anwender ermöglichen, seine Berechnungen vom Papier in den Computer zu verlegen. Es stellt dem Anwender den Rahmen zur Verfügung, der zum Erstellen eines »Arbeitsblattes« (englisch als »spreadsheet« bezeichnet) benötigt wird. Durch dieses offene Konzept läßt sich die Anwendung nicht auf ein bestimmtes Gebiet beschränken. Der Aufbau eines Spreadsheets gliedert sich in Spalten und Zeilen. Jeweils ein Feld daraus kann einen festen Wert, einen Text oder eine Formel enthalten. Dabei können Werte und Formeln beliebig untereinander verknüpft werden. Alle Ausgaben können formatiert werden. Was aber ist Multiplan und wozu braucht man es? Multiplan soll dem Anwender den Papierkram ersetzen, der zwangsläufig anfällt, wenn mehr oder weniger umfangreiche Kalkulationen durchgeführt werden sollen. Da das ganze im Computer stattfindet, können Berechnungen oder Änderungen sofort durchgeführt werden, und ihre Auswirkungen sind sofort sichtbar. Darin liegt der große Vorteil von Multiplan: Man kann mit den Werten experimentieren (lassen), bis das errechenbare Optimum gefunden ist. Zu diesem Zweck stellt das Programm neben allen mathematischen Rechenarten auch logische Verknüpfungen und statistische Operationen zur Verfügung. Ebenfalls enthalten sind Iterationen, die schrittweise Annäherung an eine Lösung. Dazu ein Beispiel: Ein Vertreter erhält 10% des Nettogewinns als Erfolgsprämie ausgezahlt. Angenommen, es werden 1000 Mark als Bruttogewinn, also ohne Abzug der Prämie erwirtschaftet. Das Problem dabei ist, daß die 10% vom Bruttogewinn einfach 100 Mark wären; dann hätte man aber 900 Mark Nettogewinn. Vom Nettogewinn jedoch sollte die Prämie berechnet werden. Also 90 Mark? Nein, dann wären wir bei 910 Mark Nettogewinn. Sie merken schon, wo das hinführt. Man muß ein paarmal im Kreis herum rechnen, bis das Ergebnis hinreichend genau ist (Das Ergebnis ist genau 1000/11; eine periodische Dezimalzahl.). Derartige Berechnungen fallen in den Bereich der Iterationen, die Multiplan für den Anwender übernimmt. Natürlich ist das ein Minimalbeispiel. Multiplan vearbeitet auch größere und dementsprechend komplexer aufgebaute Berechnungen. Damit man den Überblick nicht verliert, können zur Übersicht und zur Benutzerführung Texte in das Arbeitsblatt eingebunden werden, oder Tabellen nach bestimmten Kriterien sortiert werden. Auf einem Drucker ausgegeben sieht das Ganze dann übersichtlich und entsprechend professionell aus. Um Ihnen einen Überblick zu ermöglichen, haben wir in der Tabelle 1 die Befehlsmenüs und die Funktionen von Multiplan zusammengestellt.

Tips zu Multiplan

Das Programm ist auf zwei Disketten im 1541-Format enthalten, damit auch die Besitzer einer 1570/1571 dieses Programm nutzen können. Vor dem ersten Gebrauch erstellt man sich nach Anleitung eine Arbeitsdiskette im eigenen Diskettenformat. (Von der Verwendung der 1541 ist dabei abzusehen, da die Nachladezeiten sonst sehr lang sind). Das Kopieren dauert zwar recht lange, und danach weiß man was ein Discjockey leisten muß, doch dann kann man die Originaldisketten getrost an einen sicheren Ort weglegen.

Eines der Dienstprogramme auf der Diskette nennt sich »INSTALL«, und sollte vom Anfänger gemieden werden. »INSTALL« ist ein Programm, mit dem man verschiedene Grundeinstellungen verändern kann. Multiplan muß vom Anwender nicht behandelt werden - vor allem nicht, wenn man zum ersten Mal mit Multiplan auf dem C 128 arbeitet. Sonst kann es vorkommen, daß Sie auf Ihrem 80-Zeichen-Monitor nur ein 40 Zeichen breites Arbeitsblatt sehen oder Ihre Tastatur mit seltsamen Steuerzeichen belegt ist, etc. Eventuelle Änderungen sollte man erst mit wachsender Erfahrung auf einer Testdiskette anbringen.

Hat man seine Arbeitsdiskette erstellt, dann betätigt man den Reset-Taster des C 128 bei eingelegter Diskette und aktiviertem 80-Zeichen-Bildschirm. Nach dem Booten des CP/M-Systems sollte man, wenn möglich den (Farb-)Monitor auf »grün« umschalten. Durch die Tastenkombination »CONTROL 6« stellen Sie die grüne Schriftfarbe ein. Dann wählt man über »SETUP« und nachfolgendem »G« den deutschen Zeichensatz mit entsprechender Tastaturbelegung. Die Druckeranfrage beantwortet man seinem Drucker entsprechend. Die möglichen Drucker reichen dabei vom Commodore-Drucker (seriell) bis zu Druckern mit Centronics-Schnittstelle (über den User Port). Erst jetzt laden Sie (endlich!) Multiplan mit »MP«. So, und jetzt wäre es am besten, wenn man das Laufwerk versiegeln könnte. Würde man bei laufendem Programm die Diskette wechseln, dann stürzt Multiplan hoffnungslos ab. Die Diskette darf

Bild 2. Das Hilfe-Menü

Bild 3. Die Syntax und die Beschreibung einzelner Punkte wird deutlich.

nur nach einer Aufforderung durch das Programm gewechselt werden!

Ihr Bildschirm würde jetzt wie in Bild 1 aussehen. Das ist das leere Arbeitsblatt. Zu diesem Zeitpunkt sollten Sie sich einmal mit dem Cursor vertraut machen. Die Möglichkeiten aller Cursorbewegungen auf dem C 128 sind in Tabelle 2 aufgeführt. Leider verfügt Multiplan weder über eine Tasten-Wiederhol-Funktion noch über einen Tastaturpuffer. Der Computer nimmt die nächste Taste erst dann an, wenn er sich bis zur nächsten Eingabe durchgearbeitet hat. Dadurch wird das flüssige Arbeiten bei höherer Schreibgeschwindigkeit leider etwas gehemmt. Größere Distanzen auf dem Bildschirm sollten mit »Gehezu« überwunden werden.

Wenn man mit Multiplan ein wenig experimentiert hat (zum Beispiel nach der Anleitung im Handbuch), wird man feststellen, daß die Suche nach der Bedeutung oder Syntax einzelner Befehle im Handbuch recht lästig sein kann. Aus diesem Grund enthält Multiplan ein Hilfs-File (Bild 2). In der Befehlszeile wird es durch Anwählen von »Hilfe« aktiviert – doch das allein wäre nichts besonderes. Befindet man sich bei Multiplan irgendwo in einem Menü (außer bei der Texteingabe), wird durch Tastendruck auf das Fragezeichen der dem Menüpunkt entsprechende Hilfstext auf dem Bildschirm dargestellt (Bild 3). Wenn die Eingabe einer Position verlangt wird (Abfrage »Felder:«) muß man nicht unbedingt Zeile und Spalte eingeben, man kann den Feldzeiger dazu benutzen. Multiplan setzt dann selbständig die richtigen Koordinaten relativ zum Ausgangspunkt ein. Die relative Angabe kann dann mittels <Control G> oder »§« in eine absolute umgewandelt werden. Weitere Tips sind im Handbuch beschrieben, man sollte sich die einmal ansehen.

Für wen ist Multiplan ?

Bereits beim Öffnen der Verpackung fällt das umfangreiche Handbuch zu Multiplan auf. Es dient nicht nur der Unterstützung des Anwenders, anhand des Handbuches lernt man mit Multiplan umzugehen. Englischkenntnisse sind dazu nicht nötig, das Programm und die Beschreibung sind ausschließlich in deutsch gehalten. Multiplan auf dem C 128 erlaubt auch einem noch nicht so tief in das Computergeschehen vorgedrungenen Anwender sich schnell in die einfachen Befehle und unkomplizierten Menüs einzuarbeiten. Die professionelle Anwendung des C 128 wird von Multiplan vollständig unterstützt. Die träge Tastaturabarbeitung wird meist nur vom Schreibmaschinenprofi als störend empfunden, und wird von der Vielseitigkeit des Programms mehr als ausgeglichen. Die Nachladezeiten der Menüs sind bei der 1570/1571 durchaus erträglich, von der Verwendung einer 1541 ist abzusehen. Multiplan kann natürlich auch zwei Laufwerke bedienen. Man kann somit Programm- und Datendiskette voneinander trennen. Die etwa 14 KByte freier Arbeitsspeicher sind der Tribut an CP/M, das nach wie vor fast den halben Speicher des C 128 belegt. Für normal große Berechnungen wie zum Beispiel Umsatzplanung und -analyse, Produktionsplanung oder der Auswertung von Wettkämpfen ist das ausreichend.

(og)

Feldzeiger (das invertierte Feld links oben):
Einzelschritte mit den oberen Cursortasten, oder:

hoch:	^E
runter:	^X
links:	^S
rechts	^D

Seitenweise blättern:
vor der Cursorbewegung
^R
eingeben.
Feldzeiger auf das erste belegte Feld (Home):
^Q
Feldzeiger auf das letzte belegte oder formatierte Feld:
^Z
Feldzeiger auf das nächste belegte Feld:
^J
Befehlszeiger (das invertierte Feld in der Befehlszeile unten):

vor:	Leertaste
zurück:	DEL-Taste

es ist jedoch einfacher jeden Befehl durch Eingabe seines Anfangsbuchstabens zu aktivieren.
Bewegen der Schreibmarke im Befehlsmenü
TAB
Bewegen des Cursors im Menü »Verändern« (eines Textes oder einer Formel):

links	^K
rechts	^L
wortweise	
links	^O
rechts	^P
Zeilenende	TAB

Zurück ins Hauptmenü (zum Beispiel bei Anwahl eines falschen Menüs)
ESC oder RUN/STOP
Arbeitsblatt löschen:
Gut versteckt im Menü »Übertragen«,»Bildschirmlöschen«

Anmerkung: »^« bedeutet, daß die nachfolgende Taste zusammen mit CONTROL gedrückt wird.

Befehle und Funktionen von Multiplan

AUSSCHNITT ermöglicht die gleichzeitige Betrachtung verschiedener Bereiche einer Tabelle auf dem Bildschirm.
AUSSCHNITT TEILEN teilt den Bildschirm in bis zu acht »Fenster« auf.
AUSSCHNITT TEILEN WAAGRECHT
AUSSCHNITT TEILEN SENKRECHT
AUSSCHNITT TEILEN BEZEICHNUNG waagrecht und senkrecht für Zeilen- und Spaltenüberschriften. Die Grenze der Teilfenster ist jeweils durch Feldadressen oder Cursor-Positionierung anzugeben.
AUSSCHNITT UMRAHMUNG erlaubt die Einrahmung und damit die optische Hervorhebung eines oder mehrerer Bildfenster.
AUSSCHNITT LÖSCHEN macht die Teilung des Bildschirmes rückgängig (einzeln pro Fenster).
AUSSCHNITT VERBINDEN synchronisiert zwei oder mehrere Bildfenster.
BEWEGEN verschiebt Zeilen oder Spalten an eine andere Stelle in der Tabelle.
BEWEGEN ZEILEN verschiebt eine oder mehrere Zeilen an eine andere Stelle in der Tabelle.
BEWEGEN SPALTEN verschiebt eine oder mehrere Spalten an eine andere Stelle in der Tabelle.
DRUCK steuert die Ausgabe einer Tabelle auf einen Drucker oder auf eine Ausgabedatei.
DRUCK DRUCKER startet die Ausgabe auf einen Drucker.
DRUCK PLATTE/DISKETTE startet die Ausgabe in eine Ausgabedatei (zum Beispiel ASCII-kompatibel).
DRUCK RANDBEGRENZUNG setzt die Druckformatierung.
LINKS linker Rand (Anzahl Zeichen)
OBEN oberer Rand (Anzahl Zeilen)
DRUCKBREITE (Zeichen/Zeile)
DRUCKLÄNGE (Zeilen/Seite)
SEITENLÄNGE (Zeilen/Seite)
DRUCK OPTIONEN setzt einige Druckoptionen.
BEREICH legt den Bereich der Tabelle fest, der gedruckt werden soll.
STEUERZEICHEN erlaubt die Eingabe von Steuerzeichen für die Druckeranpassung, zum Beispiel für spezielle Schriftarten oder Schriftbilder.
FORMELN erlaubt das Ausdrucken aller Rechenformeln einer Tabelle.
Z/S-NUMMERN erlaubt das Drucken der Tabellen mit den Zeilen- und Spaltennummern.
EINFÜGEN fügt Zeilen oder Spalten ein.
EINFÜGEN ZEILE fügt eine festzulegende Anzahl von Zeilen in eine Tabelle ein. Wahlweise kann die Einfügung auf einen bestimmten Spaltenbereich begrenzt werden.
EINFÜGEN SPALTE fügt eine festzulegende Anzahl von Spalten in eine Tabelle ein. Wahlweise kann die Einfügung auf einen gewünschten Zeilenbereich begrenzt werden.
FORMAT formatiert Felder und/oder deren Inhalt.
FORMAT FELDER formatiert die angegebenen Felder.
AUSRICHTUNG Ausrichtung des Feldinhalts im Feld.
Stnd Standard: Ausrichtung wie im FORMAT STANDARD-Kommando (siehe dort).
Mitte positioniert den Feldinhalt in die Mitte des Feldes.
Norm positioniert Text linksbündig und Zahlenwerte rechtsbündig. Dies ist zugleich das von Multiplan vorab eingestellte STANDARD FORMAT.
Links positioniert jeden Feldinhalt linksbündig.
Rechts positioniert den Feldinhalt rechtsbündig.
FORMATCODE formatiert den Feldinhalt.
Stnd Standardformatierung wie im STANDARD-Kommando spezifiziert (siehe dort).
Zusamm (zusammen) gestattet das Fortschreiben eines langen Textes über die Feldgrenze hinweg in die benachbarten Felder.
E_form Exponentenschreibweise. Zahlenwerte werden als Potenz von 10 dargestellt.
Fest Festkommadarstellung. Die Anzahl der Dezimalstellen wird im Feld: »Dez-Stellen« festgelegt.
Norm zeigt Zahlenwerte in der für die jeweilige Zahl sinnvollsten Formatierung. Dies ist gleichzeitig das von Multiplan vorab gewählte STANDARD-Format.
Ganz zeigt nur den ganzzahligen, gerundeten Teil eines Wertes.
DM zeigt Zahlenwerte mit einem DM-Zeichen unmittelbar nach dem Betrag. Negative Werte werden in Klammern gesetzt.
* Balkengrafik: setzt Zahlenwerte in eine dem Wert proportionale Anzahl Sterne um.
% Prozent: multipliziert einen Zahlenwert mit 100 und setzt ein %-Zeichen hinter die Zahl.
– Keine Formatänderung
FORMAT STANDARD setzt das »Standard«-Format fest. Die mit diesem Befehl gewählte Formatierung wird später von Multiplan immer dann verwendet, wenn keine andere Formatierung ausdrücklich verlangt wird.
FORMAT STANDARD FELDER setzt das Standardformat für den Feldinhalt. Alle unter »Ausrichtung« und »Formatcode« oben aufgeführten Möglichkeiten der Formatierung können als Standard festgelegt werden.
FORMAT STANDARD BREITE_DER_SPALTEN setzt den Standardwert für die Länge aller Felder. Ursprünglich ist die Länge von Multiplan mit 10 Zeichen pro Feld festgelegt.
FORMAT OPTIONEN erlaubt die wahlweise Darstellung der Felder:
Tausenderpunkte nach jeweils drei Zahlenwerten
Formeln: mit den eingegebenen Formeln statt deren Werten. Text wird in Anführungszeichen gesetzt. Eingegebene Zahlenwerte werden unverändert gezeigt. Bei Darstellung der Formeln wird die Länge aller Felder automatisch verdoppelt.
FORMAT_BREITE_DER_SPALTE legt die Länge aller Felder fest, die von der Standardlänge abweichen sollen. Die Feldlänge kann minimal drei Zeichen und maximal 31 Zeichen betragen.
GEHEZU setzt den Cursor direkt in das gewünschte Feld.
GEHEZU NAME setzt den Cursor in das mit Namen genannte Feld. Wenn mehrere Felder denselben Namen führen, wird der Cursor in das erste Feld mit diesem Namen gesetzt, das heißt nach »links oben«.
GEHEZU ZEILE_SPALTE setzt den Cursor in das mit Zeilen- und Spaltennummer angegebene Feld.
GEHEZU AUSSCHNITT setzt den Cursor in einen anderen Bildausschnitt und in diesem in das mit Zeilen- und Spaltennummer angegebene Feld.
HILFE ruft Erklärungen zu allen Multiplan-Befehlen und -Funktionen auf. Zusätzlich wird eine Zeile mit typischen Anwendungsproblemen und deren Lösung gezeigt.
KOPIE kopiert den Inhalt eines Feldes oder Feldbereiches in ein anderes Feld beziehungsweise in einen anderen Bereich.
KOPIE RECHTS kopiert beziehungsweise vervielfältigt Felder oder Spalten nach rechts. Anzugeben ist die Anzahl der Kopien und das/die Feld(er), die kopiert werden sollen (Beginn bei:).
KOPIE NACH UNTEN kopiert beziehungsweise vervielfältigt Felder oder Zeilen nach unten. Anzugeben ist die Anzahl der Kopien und das/die Feld(er), die kopiert werden sollen (Beginn bei:).
KOPIE VON kopiert den Inhalt eines oder mehrerer Felder in einen beliebigen anderen Bereich.
LÖSCHEN löscht eine oder mehrere Zeilen/Spalten.
LÖSCHEN ZEILE löscht eine zu bestimmende Anzahl von Zeilen. Bei Bedarf kann das Löschen auf einen bestimmten Spaltenbereich begrenzt werden (von Spalte...bis Spalte).
LÖSCHEN SPALTE löscht eine zu bestimmende Anzahl von Spalten. Das Löschen kann auf einen bestimmten Zeilenbereich begrenzt werden (von Zeile...bis Zeile).
NAME vergibt beliebige Namen an eines oder mehrere Felder. Diese Namen können anschließend genauso wie Feldadressen verwendet werden.
ORDNEN sortiert die Zeilen einer Tabelle.
Als Sortierbegriff kann der Inhalt einer beliebigen Spalte verwendet werden. Das Sortieren kann auf einen bestimmten Bereich von Zeilen begrenzt werden. Auf- oder absteigende Sortierordnung kann festgelegt werden.
QUIT beendet Multiplan und gibt die Kontrolle an das Betriebssystem zurück.
RADIEREN löscht den Inhalt des angegebenen Feldes oder Feldbereiches.
SCHUTZ schützt Feldinhalte gegen unbeabsichtigtes Überschreiben.
SCHUTZ FELDER schützt oder hebt den Schutz auf für die zu benennenden Felder.
SCHUTZ RECHENFORMELN schützt pauschal alle Felder, die Formeln oder Text beinhalten. Felder, die nur Zahlenwerte enthalten, bleiben ungeschützt.
TEXT erlaubt die Eingabe von Text in das Feld, in dem sich der Cursor befindet. Nach Betätigung der Cursor-Taste (nicht der RETURN-Taste) kann die Texteingabe fortgesetzt werden, ohne erneut Text einzugeben.



ÜBERTRAGEN erlaubt die Manipulation der ganzen Tabelle.
ÜBERTRAGEN LADEN lädt eine gespeicherte Tabelle vom externen Speicher (zum Beispiel von der Diskette).
ÜBERTRAGEN SPEICHERN sichert die augenblicklich in Bearbeitung befindliche Tabelle auf dem externen Speichermedium. Multiplan schlägt hierfür einen Dateinamen vor (der aber auch frei gewählt werden kann).
ÜBERTRAGEN BILDSCHIRMLÖSCHEN löscht die augenblicklich auf dem Bildschirm befindliche Tabelle.
ÜBERTRAGEN DATEILÖSCHEN löscht eine Datei vom externen Speicher.
ÜBERTRAGEN OPTIONEN spezifiziert das Dateiformat für Datenaustausch mit anderen Programmen.
ÜBERTRAGEN UMBENENNEN erlaubt das Ändern eines Dateinamens. Dabei wird die Kopplung an externe, primäre und sekundäre Arbeitsblätter automatisch wiederhergestellt.
VERÄNDERN holt den Inhalt des angesprochenen Feldes in die Kommandozeile und erlaubt Änderungen am Feldinhalt vorzunehmen, ohne den gesamten Inhalt dabei zu löschen. Text wird dabei in Anführungszeichen gesetzt.
WERT erlaubt die Eingabe von Zahlenwerten und Formeln in das Feld, in dem sich der Cursor befindet. Zahlenwerte können auch direkt ohne WERT eingegeben werden.
XTERN steuert die Verknüpfung mehrerer Tabellen.
XTERN KOPIE kopiert Daten aus externen Tabellen in die augenblicklich aktive Tabelle. Die Daten (einzelne oder in Gruppen) können mit Namen gekennzeichnet sein. Wahlweise können die Daten schon dann kopiert werden, wenn die zu bearbeitende Tabelle von der Diskette geladen wird.
XTERN LISTE zeigt alle Tabellen an, die mit der aktiven Tabelle über XTERN KOPIE verknüpft sind.
XTERN USE erlaubt die Weitergabe aller XTERN-Verknüpfungen an andere Tabellen gleichen Aufbaus. So kann ein einziges System von gekoppelten Tabellen für mehrere Planvarianten verwendet werden.
ZUSÄTZE erlaubt die wahlweise Verwendung einiger Optionen.
SOFORT RECHNEN wählt zwischen automatisch und manuell ausgelöster Neuberechnung des Arbeitsblattes. Die automatische Neuberechnung erfolgt nach jeder Eingabe eines Zahlenwertes.
ALARM_AUS schaltet den akustischen Warnton aus/ein.
ITERATION erlaubt die iterative Lösung von Problemen.

Numerische Operatoren und Funktionen

+	Addition
-	Subtraktion
*	Multiplikation
/	Division
^	Potenzierung
%	Prozentwert (=/100)
&	Aneinanderreihung von Text

ABS(N) Absolutwert der Zahl N.
ANZAHL(LISTE) zählt alle Felder einer Liste, deren Inhalt ein Zahlenwert ist. Felder, die Text enthalten und Leerfelder werden nicht gezält. Die Liste kann sich über Zeilen und/oder Spalten erstrecken und muß nicht zusammenhängend sein.
ARCTAN(N) Arcus Tangens von (N). N ist ein Winkel im Bogenmaß.
BARWERT(Zins;Liste) ermittelt den Gegenwartswert des künftigen Rückflusses aus einer Kapitalanlage. »Zins« ist die angenommene Verzinsung (Dezimalzahl) und »Liste« die Reihe der in den einzelnen Zeiteinheiten erwarteten Rückflüsse.
COS(N) Cosinus des Winkels N. Der Winkel N ist im Bogenmaß anzugeben.
DELTA( ) setzt die Endebedingung für Iterationen. Eine Iteration wird als gelöst beendet, wenn sich zwei aufeinanderfolgende Näherungslösungen um weniger als den Betrag DELTA( ) unterscheiden.
DMARK(N;S) wandelt einen Zahlenwert in Text um ein »DM« hinter die Zahl. Mit S kann die Anzahl der Dezimalstellen nach dem Komma angegeben werden. Negative Werte werden in Klammern gesetzt.
EXP(N) Exponentialfunktion auf der Basis e (=2,7182818...). Dies ist die inverse Funktion zu LN(N).
FEST(N;Stellen) wandelt einen Zahlenwert N in Text mit vorgegebener Anzahl von Dezimalstellen um (siehe auch Funktion WERT(T)).
GANZZAHL(N) behält den ganzzahligen Teil eines Zahlenwertes und entfernt alle Dezimalstellen.
INDEX(Bereich;Lage) liefert den Zahlenwert eines Feldes, das mit »Lage« aus einem rechteckigen Feldbereich selektiert wurde. Mit dieser Funktion ist es möglich, zum Beispiel die Zahlen einer Spalte in eine Zeile zu transferieren oder umgekehrt.
LÄNGE(T) zählt die Anzahl der Zeichen eines Textes (T).
LN(N) ergibt den natürlichen Logarithmus der Zahl N. Dies ist die inverse Funktion zu EXP(N).
LOG10(N) ergibt den Logarithmus der Zahl N auf der Basis 10.
MAX(Liste) ermittelt den Maximalwert einer Zahlenreihe. »Liste« kann eine Zeile, eine Spalte, ein Bereich oder eine lose Aneinanderreihung von Zahlenwerten sein.
MIN(Liste) ermittelt den Minimalwert einer Zahlenreihe. »Liste« kann eine Zeile, eine Spalte, ein Bereich oder eine lose Aneinanderreihung von Zahlenwerten sein.
MITTELW(Liste) Durchschnittswert einer Zeile von Zahlen. Dies kann eine Zeile zusammenhängender oder loser Zahlen sein. Der Bereich kann beliebig groß sein.
NV( ) liefert den Fehlerwert NV. Diese Funktion kann benutzt werden, um Felder zu markieren, für die später noch Zahleneingaben benötigt werden.
PI( ) bringt die Zahl PI zur Anzeige: PI = 3,141592653...
REST(N;M) ergibt den »Rest« aus der Division N/M.
RUNDEN(N;Stellen) rundet einen Zahlenwert N auf so viele Dezimalstellen wie mit »Stellen« angegeben. Mit negativen Zahlen als »Stellen« kann auf 10, 100, 1000 usw. gerundet werden.
SIN(N) bringt den SINUS des Winkels N, wobei N im Bogenmaß anzugeben ist.
SPALTE( ) liefert die Nummer der Spalte, in der die Funktion steht, als Zahlenwert. Damit kann die Spaltennummer abgefragt und innerhalb von Formeln weiterverarbeitet werden.
STABW(Liste) ermittelt die Standardabweichung der Werte einer Zahlenreihe. »Liste« kann eine Zeile, eine Spalte, ein Bereich oder eine lose Aneinanderreihung von Zahlen sein.
SUCHEN(N;Bereich) sucht in der ersten Spalte oder Zeile des »Bereichs« nach dem Zahlenwert N. Ist der Wert gefunden, wird in dessen Zeile oder Spalte der Feldinhalt der letzten Spalte oder Zeile zur Anzeige gebracht. Mit SUCHEN sind Zugriffe auf einzelne Daten einer Tabelle möglich.
SUMME(Liste) bildet die Summe aller Zahlenwerte einer »Liste«. »Liste« kann eine Zeile, eine Spalte, ein Bereich oder eine lose Aneinanderreihung von Zahlen sein.
TAN(N) ergibt den TANGENS des Winkels N, wobei N im Bogenmaß anzugeben ist.
TEIL(T;Beginn;Länge) liefert eine Zeichenfolge aus einem Text T. Die Zeichenfolge hat die Länge »Länge« und beginnt mit dem Zeichen, das an der »Beginn«-Position innerhalb des Textes steht. »Beginn« und »Länge« werden als Nummern angegeben.
VORZEICHEN(N) untersucht das Vorzeichen einer Zahl und bringt davon abhängig einen Zahlenwert:

bei positivem Vorzeichen:	+1
bei negativem Vorzeichen:	-1
bei N = 0:	0

WERT(T) wandelt einen Text T in einen Zahlenwert um. Der Text muß so beschaffen sein, daß die Umwandlung einen syntaktisch korrekten Zahlenwert ergibt.
WIEDERHOLEN(T;Anzahl) wiederholt einen Text T so oft, wie mit der Zahl »Anzahl« angegeben wird. WIEDERHOLEN kann verwendet werden, um Zahlenwerte in entsprechend lange Balkengrafiken umzusetzen.
WURZEL(N) ermittelt die Quadratwurzel der Zahl N
ZÄHLER( ) bringt bei Iterationen einen Zähler für die Anzahl der Iterationsschritte zur Anzeige. Abhängig vom Zählerstand können zum Beispiel Verzweigungen des Rechengangs veranlaßt werden.
ZEILE( ) liefert die Nummer der Zeile, in der die Formel steht. Damit kann die Zeilennummer abgefragt und als Zahlenwert weiterverarbeitet werden.

Logische Funktionen

FALSCH( ) liefert den logischen Wert »falsch«.
ISTFEHL(Wert) ergibt den logischen Wert »wahr«, wenn in dem abgefragten Feld ein Fehlerwert steht.
ISTNV(Wert) ergibt den logischen Wert »wahr«, wenn in dem abgefragten Feld der Fehler NV! steht.
NICHT(Logisch) ergibt den entgegengesetzten logischen Wert des Arguments.
ODER(1.Bed.;2.Bed.) ergibt den logischen Wert »wahr«, wenn eines der Argumente den logischen Wert »wahr« hat.
UND(1.Bed.;2.Bed.) ergibt den logischen Wert »wahr«, wenn alle Argumente von UND den logischen Wert »wahr« haben.
WAHR( ) liefert den logischen Wert »wahr«.
WENN(Logisch;Dannwert;Sonstwert testet einen logischen Wert »Logisch«. Ist dieser »wahr«, dann wird die Funktion »Dannwert« ausgeführt, andernfalls die Funktion »Sonstwert«.

Quelle: Albrecht, Peter: Multiplan für den Commodore 128 PC. Professionelles Planen und Kalkulieren mit dem Commodore 128 PC. Markt & Technik Verlag 1985, S. 191 ff.

dBase II – die Super-Datenbank auf dem C 128

Komfort und Flexibilität – eine Datenbank wie auf einem Großcomputer! Das gibt es jetzt auch für den C128. dBase II ist wohl das beste Datenbank-System für Personal Computer. Ein ausführlicher Bericht zeigt, was dieses Programm leistet.

Sie glauben gar nicht, was man mit einer Datenbank alles machen kann! Vorausgesetzt, man verfügt über das richtige Werkzeug.

Beispiel: die Redaktion einer Computerzeitung.

Welche Computer und Monitore, Drucker, Diskettenstationen, Plotter, Interfaces, welche Software und welche Bücher sind wann eingegangen? Welcher Redakteur hat sie zur Zeit? Wann muß was zurückgeschickt werden? Was ist gekauft und was geliehen? Wo sind Adapter und einzelne Druckerkabel, die sehr gerne verlorengehen? Diese Fragen kann dBase II im Einsatz beantworten.

Nehmen wir das Beispiel Vereinsverwaltung. Über dBase II kann man alle Informationen über die Mitglieder verwalten, Beitragsvordrucke schreiben (das Programm weiß: Jugendliche zahlen die Hälfte vom Vater, pro Familie wird nur ein Vordruck verschickt und anderes mehr).

Beispiel Lagerverwaltung: Welche Artikel sind in welcher Stückzahl im Lager? Wieviel Teile wurden verkauft und was muß bestellt werden? Mit dBase II wird das Lager problemlos verwaltet, Bestellungen und ausstehende Rechnungen können automatisch ausgedruckt werden.

Das Datenbank-System dBase II, das dies alles leistet, ist in einer CP/M Plus-Version nun auch für den C 128 verfügbar. dBase II ist eines der populärsten Datenbanksysteme und läuft auf 8- und 16-Bit-Mikroprozessoren. dBase II ist leicht erlernbar. Mit wenigen Befehlen kann auch ein Benutzer, der bisher mit Datenbanken und Programmieren keine Erfahrung hat, Dateien erstellen, Informationen eingeben und sie auswerten. Für Programmierer ist interessant, daß dBase II eine eigene Programmiersprache besitzt, die nicht nur sehr vielseitig ist, sondern auch praktisch. Auch kompliziertere Programme sind leicht verständlich und wegen der leistungsstarken Befehle von überraschender Kürze.

dBase II ist ein relationales Datenbanksystem. Nach einer allgemeinen Erklärung der Begriffe »Datenbanksystem« und »relationales Datenbanksystem« wird in diesem Artikel anhand eines Beispiels (Videoclip-Datei) demonstriert, wie man mit dBase II arbeitet.

Mit einem Dateiverwaltungs-Programm kann ein Benutzer eine spezifische Datei verwalten, das heißt, neue Datensätze eintragen, bereits vorhandene Datensätze ändern oder löschen. Und er kann beliebig viele verschiedene Dateien mit dem Verwaltungsprogramm aufbauen. Ein Datenbanksystem kann alles dies auch. Darüber hinaus können hier aber auch mehrere Dateien miteinander verknüpft werden. Und ein ganz wesentlicher Unterschied ist die eigene Programmiersprache, über die ein Datenbanksystem verfügt.

Diese beiden Merkmale unterscheiden ein Datenbanksystem von einer einfachen Dateiverwaltung: das Verknüpfen von Dateien und die Möglichkeit zur Programmierung immer gleicher Arbeitsabläufe.

Bei der üblichen Datenorganisation in Programmen führt jedes Programm seine eigenen Dateien. Inhalte dieser Dateien können sich für unterschiedliche Programme durchaus überschneiden. Dieses mehrfache Speichern von Daten belegt unnötigen Speicherplatz. Ein weiterer Nachteil dieser Datenorganisation tritt bei aufeinander aufbauenden Programmen auf: Daten müssen an übergeordnete Programme weitergegeben werden. Dieser Aufwand ist nicht mehr nötig, wenn alle einmal gespeicher-

Was ist eigentlich ein Datenbanksystem?

Die Programmiersprache in diesem Datenbank-System enthält nicht nur sehr mächtige Datenbankbefehle, sondern auch alle wichtigen Elemente höherer Programmiersprachen, das sind IF-THEN-ELSE-Abfragen, DO-Schleifen, Rechenoperationen und die Verwendung von Speichervariablen. dBase II setzt aber keine Programmierkenntnisse voraus! Alle Funktionen – wie zum Beispiel das Einrichten einer Datei, die Dateieingabe, das Sortieren und Ausgeben von Daten – können auch im direkten Dialog ausgeführt werden.



ten Daten allen Programmen zugänglich gemacht werden – und dies ist genau das Prinzip eines Datenbanksystems.

In einem Datenbanksystem werden also alle Daten einmal gespeichert und mit dieser Datenbank arbeiten alle Programme (Bild 1).

Darüber hinaus hat ein Datenbanksystem die Aufgabe, Informationen in geordneter Form zu verwalten und dem Benutzer zur Verfügung zu stellen. Praktisch alle Daten, die in einer Datenbank gespeichert werden, stehen untereinander in irgendeiner Beziehung. In einer Personaldatenbank beispielsweise steht der Name eines Mitarbeiters in Beziehung zu seinem Monatseinkommen. Andererseits ist der Mitarbeitername wieder verknüpft mit persönlichen Daten oder dem Namen der Abteilung. Dies ist ein ganz einfaches Beispiel. Die Beziehungen von Daten einer Datenbank zu anderen Daten aus weiteren Datenbanken können sehr komplex sein. Die Ordnung innerhalb einer Datenbank, die Datenbankstruktur, baut auf diesen Beziehungen auf. In einer Datenbank ist also festgelegt, wo welche Daten liegen, in welcher Form sie schon verknüpft sind oder verknüpft werden können.

Datenstruktur

Eine Datenbank besteht üblicherweise aus mehreren Dateien (Files). Eine solche Datei kann man sich vorstellen als Karteikasten, der eine Menge Karteikarten enthält. Die Karteikarten in der Datenbank nennt man Datensatz. Ebenso wie eine Karteikarte nach einem festen Schema beschrieben wird, das sich aus Einzelangaben zusammensetzt, hat jeder Datensatz ein festes Schema. Dieses Schema gibt eine feste Struktur vor, die für jeden Datensatz gleich ist. Jede Einzelangabe auf einem Datensatz wird als Feld bezeichnet.

Der Aufbau einer Datenbank nun noch einmal anschaulich:

Eine Datei (Bild 2) in der Datenbank enthält zum Beispiel ein Verzeichnis aller auf Videoband aufgezeichneten Clips. Die Angaben in der Datei können als Liste (Bild 3) ausgegeben werden.

Jede Zeile entspricht einem Datensatz (Bild 4). Jeder dieser Datensätze ist nach einem festen Schema aufgebaut. Das Schema der Datensätze dieses Beispiels enthält vier Felder – »Titel«, »Interpret«, »Bandnr« und »Stereo«. Die Feldtypen – das sind die Arten der Datenfelder – sind in den Feldern »Titel« (Bild 5) und »Interpret« alphanumerische Zeichenfolgen, beim Feld »Bandnr« eine numerische oder Zahlenfolge. Das Feld »Stereo« ist ein logisches Feld, das nur y oder n, ja oder nein enthalten darf. Die Länge von Feldern wird beim Anlegen der Datei festgelegt. Felder sind in allen Datensätzen gleich

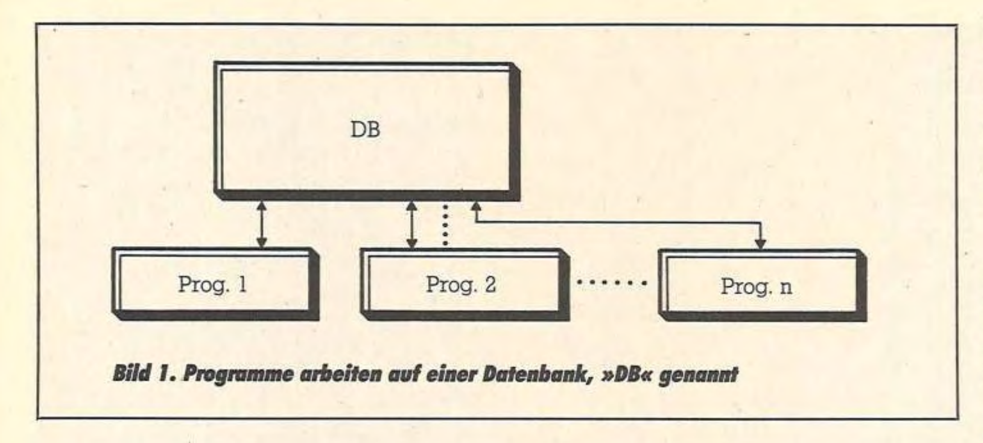
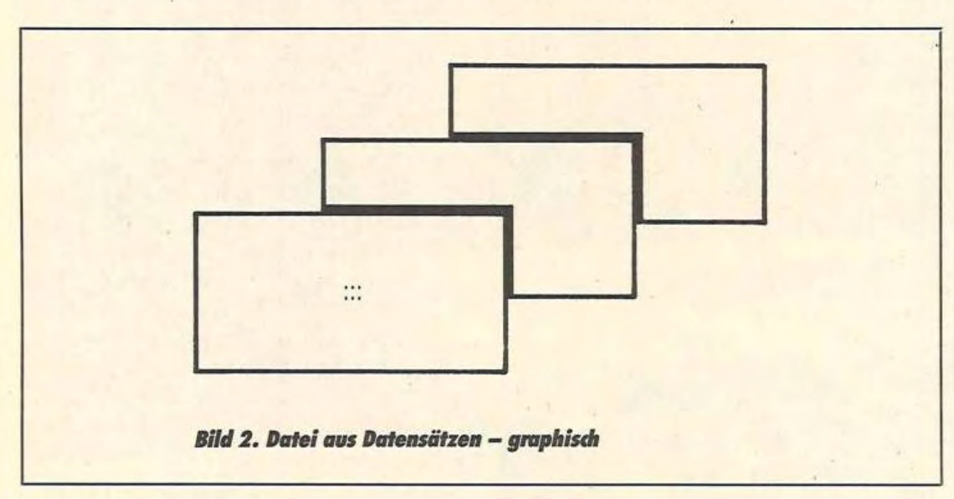


Bild 1. Programme arbeiten auf einer Datenbank, »DB« genannt


Bild 2. Datei aus Datensätzen – graphisch

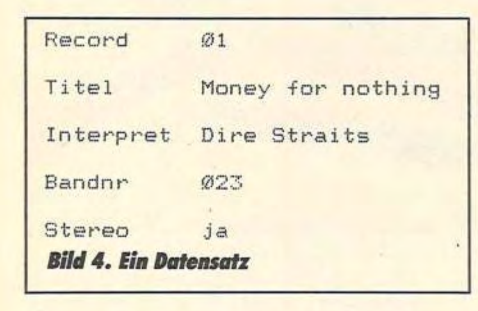


Bild 4. Ein Datensatz

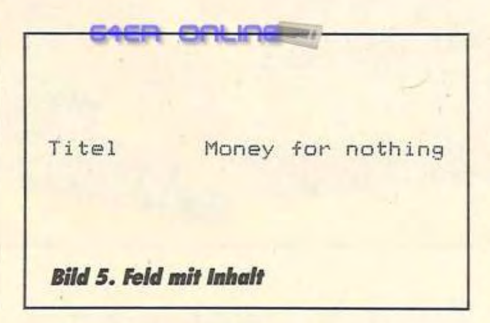


Bild 5. Feld mit Inhalt

Record	Titel	Interpret	Bandnr	Stereo
Ø1	Money for nothing	Dire Straits	Ø23	ja
Ø2	Love is the seventh wave	Sting	Ø25	ja
Ø3	Uptown girl	Billy Joel	Ø14	ja

Bild 3. Eine Datei

lang. Nicht beschriebene Stellen werden mit Leerzeichen aufgefüllt.

Was ist ein relationales Datenbanksystem?

Unter Relation versteht man eine Menge von Paaren oder Tupeln (drei oder mehr) aus gleichartigen Elementen, zwischen denen eine bestimmte Beziehung besteht.

In einer relationalen Datenbank sind diese Elemente die Datensätze. Im Bild 3 ist eine solche Relation dargestellt. Zwischen den Elementen »Uptown girl«, »Billy Joel«, »014« und »ja« – dem Inhalt des Datensatzes 3 – besteht beispielsweise eine (von vielen möglichen!) Beziehung: sie sind in demselben Datensatz enthalten.

In einer relationalen Datenbank werden die Daten als eine Folge von zweidimensionalen Tabellen gespeichert, in denen die Tabellenzeilen den einzelnen Sätzen der Datenbank und die Spalten den Feldern entsprechen. Auf diese Weise leistet ein relationales Datenbanksystem die Strukturierung der Daten, die bei anderen Formen der Dateiverwaltung vom Benutzer organisiert werden muß.

Das Strukturkonzept, das einer relationalen Datenbank zugrunde liegt, ist sehr einfach und erlaubt aufgrund dessen schnelles Definieren, Benutzen und Ändern der Datenstrukturen. Die dBase II-Befehle, die dies leisten, heißen »CREATE«, »USE« und »MODIFY STRUCTURE«. Sie werden weiter unten genau beschrieben. Darüber hinaus bietet dieses Strukturkonzept die flexibelste Art, Daten zu speichern und zu verwalten. Jedes Element kann mit jedem anderen verbunden werden, ohne daß die Beziehung vorher definiert werden mußte. Diese Verknüpfungen werden in unserem Datenbanksystem realisiert durch die drei logischen Operationen »und«, »oder« und »nicht«, die in dBase II folgendermaßen heißen: ».AND.«, ».OR.« und ».NOT.«. In der Beschreibung der Befehle »LIST« und »DISPLAY« wird darauf näher eingegangen.

Anwendungsmöglichkeiten für dBase II

Die Anwendungsbereiche von dBase II sind sehr weitgefächert. Sie reichen von klassischen Anwendungen wie Adreß- und Rechnungsverwaltung und dem Aufbau von Ausdruckformularen über die Organisation eines Bibliothekskatalogs, in dem Datenmengen verschiedener Größe im ständigen Zugriff gehalten und verändert werden müssen, bis zum Datenaustausch mit Textverarbeitungsprogrammen wie WordStar (Bild 6) und Tabellen-Kalkulationsprogrammen wie zum Beispiel Multiplan (Bild 7 und 8).

Im folgenden Abschnitt wird veranschaulicht, wie mit Hilfe von dBase II Daten verwaltet werden.

Über die vielfachen Einsatzmöglichkeiten hinaus kann dBase II so mit Serviceprogrammen ausgestattet werden, daß Benutzer damit arbeiten können, ohne die Dialogsprache von dBase II zu kennen (Bild 9). Die integrierte Programmiersprache des Datenbanksystems enthält Befehle, mit deren Hilfe Bildschirmmasken schnell und problemlos erstellt werden können. Durch solche Masken kann das Arbeiten mit einem Datenbanksystem für computerungeübte



```
.DF test3.TXT
 .RV name, vorname, strasse, ort, betrag
Herrn &vorname& &name&                                    23.August 1983
&strasse&
&ort&

Sehr geehrter Herr &name&,
wie wir leider feststellen mußten, ist unsere letzte Rechnung über
                                          DM &betrag&

offenbar noch nicht beglichen.......
```

Bild 6. WordStar-Serienbrief mit Datenübernahme aus dBase II

	1	2	3	4	5	6	7
1			Budget	Blatt 1			
2							
3			in 1000 DM				
4							
5		Januar	Februar	März	April	Mai	Juni
6	-------------	------	------	------	------	------	------
7	Gehälter	72	74	73	75	76	74
8	Pers.Nebenk.	21	23	24	25	24	25
9	Betriebsmittel	10	4	8	6	9	8
10	Dienstleist.	20	24	31	26	22	25
11	Abschreibungen	15	15	15	15	16	17
12	Sonst.Kosten	5	2	6	9	1	10
13	-------------	------	------	------	------	------	------
14	Summe	143	142	157	156	148	159

Bild 7. Multiplan-Tabelle

00001	Gehälter	72	74	73	75	76	74
00002	Pers.Nebenk.	21	23	24	25	24	25
00003	Betriebsmit.	10	4	8	6	9	8
00004	Dienstleist.	20	24	31	26	22	25
00005	Abschreib.	15	15	15	15	16	17
00006	Sonst. Kost.	5	2	6	9	1	10

Bild 8. Daten einer dBase II-Datei, die von Multiplan übernommen sind

```
Programm-Auswahl

(1) Informationen über einzelne Kunden
(2) Kundenadressen auflisten
(3) neue Kunden aufnehmen
(4) Rechnungs-Bearbeitung
(0) ENDE

Bitte wählen Sie eine Funktion
```

Bild 9. Bildschirmmaske zur Programmauswahl

```
 . create video-clip
Satzstruktur folgendermaßen eingeben
Feld         Name, Typ, Länge, Dezimalstellen
001          titel,c,25,0
002          interpret,c,12,0
003          bandnr,n,3,0
004          stereo,1,1,0
005
Daten jetzt eingeben? N
 . use video-clip
 . modify structure
```

Bild 10. Anlegen einer neuen Datei mit Satzstruktur

Benutzer wesentlich vereinfacht werden. Bild 9 zeigt das Beispiel einer solchen Bildschirmmaske. Der Benutzer kann eine der angegebenen Funktionen wählen. Die Eingabe »0« beendet das Programm. Wählt man die Funktion »1«, so werden Informationen über Kunden aufgelistet. Wie das Programm aussieht, das diese Maske erzeugt, zeigt Bild 15. Durch den dBase-Befehl »SAY« wird der Text an den angegebenen Positionen auf dem Bildschirm gedruckt. Die Positionsangabe »1,30« bedeutet: in der 1. Zeile auf der 30. Stelle auf dem Bildschirm beginnt der Text. In der CASE-Schleife wird die Antwort des Benutzers eingelesen und ausgewertet. Wie man sieht, führt das dBase-Programm je nach Eingabe (»1«, »2«, »3«, »4« oder »0«) verschiedene Aktionen aus. Auf die Eingabe »1« hin wird ein Unterprogramm namens »info2« aufgerufen, das Informationen über einzelne Kunden liefert. Das Programm »info2« ist nicht abgebildet. Auf die Eingabe »0« hin wird durch Ausführen des Befehls »RETURN« das Programm beendet.

Das Arbeiten mit dBase II

Bevor dBase II auf dem C 128 aufgerufen werden kann, wird über das mitgelieferte Hilfsprogramm SETUP der Zeichensatz festgelegt, mit dem dBase zukünftig arbeiten soll. Der Commodore 128 bietet den internationalen ASCII-Zeichensatz mit Sonderzeichen, wie eckige und geschweifte Klammern, und den deutschen DIN-Zeichensatz mit Umlauten, »ß« und dem »%«-Zeichen.

Da die deutschen Umlaute in den Datenbeständen häufig auftauchen, sollte auf die Systemfrage
GERMAN (G) OR ASCII (A)
KEYBOARD
hin der deutsche Zeichensatz gewählt werden. Damit taucht das erste Problem auf: Commodore-Drucker kennen keine deutschen Umlaute. Besser geeignet sind Epson-Drucker und Epson-kompatible Drucker wie Star SG10/SD10/SR10, Ritemann II, Citizen MSP10-MSP20, Ritemann C+/F+ oder Seikosha SP 1000/SP1300. Diese Drucker verfügen über den deutschen Zeichensatz.

Nun soll an ganz einfachen Beispielen gezeigt werden, wie Befehle in dBase II aussehen, wie sie funktionieren und welche Probleme in einem Datenbanksystem wie gelöst werden können. Damit soll denjenigen, die dBase II noch

nicht kennen, ein Gefühl dafür vermittelt werden.

dBase II wird aufgerufen durch den Befehl:
DBASE.

Das Datenbanksystem meldet sich mit dem Prompt-Punkt, einem einzelnen Punkt in einer Zeile. Damit zeigt das System an, daß Befehle eingegeben werden können. Verlassen wird dBase II durch den Befehl
QUIT.

Erstellen einer Datei und Dateneingabe.

Legen wir nun eine neue Datei an und nennen sie video_clip:
CREATE »video_clip«

In Bild 10 wird gelistet, wie das System die Satzstruktur abfragt. Die Felder werden automatisch durchnumeriert. Jedes Feld wird vom Benutzer mit einem Namen belegt. In unserem Beispiel soll das erste Feld den Titel enthalten. Deshalb wird es auch »titel« genannt. Da der Titel ein Text ist, muß das Feld vom Typ Zeichenkette sein. Für das System wird der Feldtyp Zeichenkette durch ein »c« kodiert (»c« steht für character). Als Feldlänge nehmen wir 25 Stellen an, weil die Titel wahrscheinlich nicht länger sind. Und die Anzahl der Dezimalstellen ist beim Feldtyp Zeichenkette natürlich »0«. Das nächste Feld heißt »interpret«. Die Angaben werden ebenso wie beim ersten Feld durch Komma getrennt eingegeben. Das dritte Feld soll die Bandnummer, das heißt also Zahlen, enthalten. Das Feld wird »bandnr« genannt. Der Feldtyp eines Zahlenfeldes (numerisches Feld) wird als »n« kodiert. Da es wahrscheinlich nicht mehr als 999 Bänder gibt, reicht uns die Feldlänge »3«. Dezimalstellen werden nicht benötigt. Das letzte Feld »stereo« soll nur ja oder nein enthalten. Daher hat es den Feldtyp »l«, das heißt »logisches Feld«. Ist die Eingabe der Satzstruktur beendet, so fragt das System »Daten jetzt eingeben?«. Gibt man darauf ein »y« (ja) als Antwort ein, so können die Daten direkt eingegeben werden.

Beendet werden kann die Eingabe wieder mit dem Befehl
CTRL w.

Wenn wir die Eingabe der ersten Sätze nun auf Schreibfehler überprüfen wollen, kann durch
EDIT 1
der erste Satz wieder auf den Schirm geholt und gegebenenfalls korrigiert werden.

Beendet wird das Editieren wieder durch das Kommando
CTRL w.

Sollen in eine schon existierende Datei Daten eingegeben werden, so muß der Name der Datei zuvor dem System bekanntgegeben werden. Dies geschieht mit
USE video_clip

Damit wurde die Datei video_clip dem System nun als die aktuelle Datei gemeldet. Auf ihr kann man nun alle möglichen Operationen durchführen. Auch in der Vereinbarung der Dateistruktur können Änderungen und Erweiterungen vorgenommen werden. Mit dem Kommando
MODIFY STRUCTURE
kann die Dateistruktur editiert und verändert werden, um zum Beispiel die Titellänge zu erhöhen oder um ein weiteres Feld hinzuzufügen. Die bisher eingegebenen Daten gehen bei Ausführung dieses Kommandos allerdings verloren. Daher muß man die Daten zuvor in eine zweite Datei kopieren (mit dem Befehl COPY), von der man sie später in die geänderte Ursprungsdatei zurückschreiben kann. Mit dem Kommando
CTRL w
wird der MODIFY-Modus wieder verlassen. Um nun Daten einzugeben, benötigt man das Kommando
APPEND

Es liefert einen leeren Datensatz auf den Schirm und erwartet die Eingabe eines Datensatzes. Wenn dieser beschrieben ist, blättert das System weiter und liefert den nächsten leeren Datensatz.

Wir haben bisher eine neue Datei angelegt (mit »CREATE«) und unsere Daten eingegeben (durch »APPEND«). Nun kann man auflisten lassen, welche Informationen bisher eingegeben wurden:
LIST STRUCTURE listet die Dateistruktur (Bild 11),
LIST listet den Dateiinhalt komplett (Bild 12),
LIST titel,interpret listet nur die eingegebenen Titel und Interpreten.

Über die drei logischen Grundfunktionen
.and. (und, das heißt: sowohl ... als auch),
.or. (oder, das heißt entweder ... oder) und
.not. (nicht)
können verknüpfte Suchbedingungen erzeugt werden. Damit können ganz bestimmte Daten gesucht und gelistet werden.

Werden zum Beispiel in einer Kundendatei alle Personen namens Meier gesucht, formuliert man die entsprechende Bedingung so:
LIST FOR name = 'Meier'.

Ist man sich über die Schreibweise des Namens nicht ganz sicher, so gibt man zwei Schreibweisen an und verbindet sie mit »oder«.

Das sieht dann so aus:
LIST FOR name = 'Meier'.OR. name = 'Meyer'

Will man nur die Mitarbeiter namens 'Meier' in München aufgelistet haben, so fragt man zusätzlich nach der Postleitzahl »8000« im Feld »plz« und fragt so ab:
FOR name = 'Meier'.AND. plz = '8000'

Ein Beispiel für eine »NOT«-Abfrage findet sich unter der Beschreibung der »WHILE«-Schleife.

Die nächsten Kommandos arbeiten auf größeren Datenmengen. Da unsere Mini-Datei bisher nur drei Sätze enthält, wird nun an einer größeren Datei, die in Bild 12 gelistet ist, weitergearbeitet.
Durch das Kommando
LIST
werden alle Feldinhalte, die einer angegebenen Bedingung genügen, gelistet. Will man sich satzweise den jeweils nächsten Datensatz anzeigen lassen, der die Bedingung erfüllt, verwendet man den folgenden Befehl:
DISPLAY FOR postltzahl = '8'
listet den nächsten Datensatz, in dem die Postleitzahl mit '8' beginnt, hier also den ersten.

```
LIST STRUCTURE
STRUKTURDATEN FÜR DATEI: TEMP.DBF
ANZAHL DER SÄTZE: 00010
DATUM DER LETZTEN AKTUALISIERUNG: 00/00/00
PRIMÄRE DATEI
FELD        NAME            TYP        LÄNGE    DEZIMALSTELLEN
001         NAME            C          020
002         ADRESSE         C          020
003         POSTLEITZAHL    C          004
004         STADT           C          020
005         TELEFON         C          010
006         BUNDESLAND      C          003
))TOTAL((                              00078
```

Bild 11. Ausgabe von LIST STRUCTURE – Liste der Dateistruktur

Den nächsten Datensatz, der die Suchbedingung erfüllt, findet man durch Eingabe des Befehls
CONTINUE

Durch Abwechseln der Befehle
DISPLAY und CONTINUE
kann man in Einzelschritten das gleiche erreichen wie oben mit dem Befehl LIST. Diese schrittweise Ausführung wird im Rahmen von Programmen interessant.

Zur gleichzeitigen Bearbeitung mehrerer Datensätze dient der Befehl
BROWSE

Mit diesem Befehl können zwanzig aufeinanderfolgende Datensätze gleichzeitig auf den Schirm geholt und bearbeitet werden. Die Datensätze der gesamten Datei können durch Cursorbewegung erreicht und so bearbeitet werden.

Beim Aufbau einer Datenbank müssen naturgemäß viele Informationen erstmals über die Tastatur in das System eingegeben werden. Dies ist zeitlich aufwendig und bietet viele Fehlermöglichkeiten. Daher sollten möglichst viele bereits vorhandene Informationen für die Einrichtung neuer Dateien genutzt werden. Umfassende Möglichkeiten dafür sind ein wesentliches Merkmal guter Datenbanken.

Durch den Befehl
APPEND FROM <dateiname>
können ausgewählte Informationen aus der aktuellen Datei in eine neue Datei übernommen werden. (Die genaue Syntax der dazu notwendigen Befehlsfolge findet sich in jedem dBase II-Handbuch – siehe Literaturliste am Ende des Artikels.)

Ein Befehl, der typische Eigenschaften einer relationalen Datenbank realisiert, ist die Anweisung
REPLACE <Feldname> WITH
<Ausdruck>

Ein beliebiger Datensatz ist im Zugriff. In diesem Datensatz wird nun der Inhalt des Datenfeldes <Feldname> ersetzt durch einen Text, eine Zahl, den Inhalt eines anderen Feldes oder das Ergebnis einer Berechnung – je nach dem <Ausdruck>.

Hätten wir beispielsweise in der aktuellen Datei ein Feld namens »datum«, so würde durch den folgenden Befehl in allen Sätzen der Datei der Inhalt dieses Feldes geändert.
REPLACE ALL datum WITH Date()
In einem Systemspeicher
Date() steht das aktuelle Datum, das beim Start von dBase II angegeben wurde. Der Inhalt des Feldes »datum« wird durch die Ausführung

```
00001 Adams, Peter    Bergstrasse 15   8000 München 2    14 12 53 BAY
00002 Camphausen,E.   Müllerstr. 7     1000 Berlin 2     84 00 33 BLN
00003 Dirschedel,F.   Magnolienweg 20  8000 München 90   97 81 13 BAY
00004 Dollinger,E.    Hanfstr.43       7432 Unterdorf    12 45    B-W
00005 Eberhart, T.    Marktstr.19      5013 Ebenhausen   34 56 23 NRW
00006 Faltmann, G.    Däumlingsweg 6   8049 Dirnbach     13 29 8  BAY
00007 Gruber,Otto     Mozartstr.29     7080 Pfaffenh.    46 28    B-W
00008 Haberer,Karl    Teufelsweg 13    8047 Schrobenh.   12 45 7  BAY
00009 Imhof,Bernh.    Gärtnerstr.9     4703 Marktdorf    123      NRW
00010 Jahn,Alois      Hühnersteig 3    6142 Oberhof      456      BAY
```

Bild 12. Liste eines kompletten Dateiinhalts

```
@  1,30  SAY  ' Kunden-Information  '
@  5,5   SAY  TRIM (vorname) +  ' ' + name
@  5,50  SAY  ' kunden-Nr. '   ' + kunummer
@  7,5   SAY  strasse
@  8,5   SAY  ort
@  12,5  SAY  ' Letzte Rechnung vom: '   ' + datum
@  15,5  SAY  ' bisheriger Umsatz:  DM '
@  17,5  SAY  ' offene Posten:      DM  '
```

Bild 13. Positionierungsbefehle für eine Bildschirmmaske

```
                      Kunden-Information

Werner Meister                                    Kunden-Nr.098
Marktplatz 3
8057 Eching


Letzte Rechnung vom:

bisheriger Umsatz: DM
Offene Posten:          DM
```

Bild 14. Bildschirmmaske

```
STORE T TO menue
DO WHILE menue ERASE
   @ 1,30 SAY 'Programm-Auswahl'
   @ 5,15 SAY '(1) Informationen über einzelne Kunden'
   @ 7,15 SAY '(2) Kundenadressen auflisten'
   @ 9,15 SAY '(3) neue Kunden aufnehmen'
   @ 11,15 SAY '(4) Rechnungs-Bearbeitung'
   @ 13,15 SAY '(0) ENDE'
   @ 19,15 SAY 'Bitte wählen Sie eine Funktion'
   WAIT TO antwort
   DO CASE
         CASE antwort = '1'
            DO info2
         CASE antwort = '2'
            USE adressen INDEX alphabet
            ERASE
            @1,1 SAY 'Kunde  Strasse  Wohnort
Kunden-Nr.'
            DISPLAY OFF ALL $(vorname,1,1)+'.',
            name,strasse,ort,kunummer;
            WAIT
         CASE antwort = '3'
            USE adressen INDEX alphabet
            APPEND
         CASE antwort = '4'
            ?
         CASE antwort = '0'
            RETURN
   ENDCASE
ENDDO
```

Bild 15. Programm für ein Bildschirm-Menü

des REPLACE-Befehls durch dieses Tagesdatum ersetzt.

Eine solche Manipulation einer kompletten Datei oder Teilen einer Datei mit einem einzigen Befehl (hier durch REPLACE ALL) ist nur bei relationalen Datenbanken möglich.

dBase II bietet über die hier gezeigten Befehle hinaus wesentlich mehr Möglichkeiten zur Dateneingabe, -manipulation und zur -auswahl. Alles aufzuzeigen, was in diesem Datenbank-System möglich ist, würde den Rahmen dieser Einführung sprengen. Es gibt mehrere empfehlenswerte Bücher über dBase II, in denen das System in seinem vollen Umfang beschrieben wird (siehe Ende des Artikels).

Sortieren, Suchen, Indizieren

Eine der wichtigsten Aufgaben einer Datenbank ist die sortierte Ausgabe. Sortiert wird eine Datei durch Ausführung des Befehls:
SORT ON <Feldname> TO <Zieldatei>

Der Sortierbegriff, nach dem sortiert wird, ist <Feldname>.

Das soll nun konkret gemacht werden. Nimmt man wieder die Datei von Bild 12, so kann man diese nach verschiedenen Kriterien sortieren. Durch:
SORT ON name TO test
werden die Datensätze in alphabetischer Reihenfolge geordnet und in die neue Datei »test« geschrieben.

Das Sortieren einer Datei ist bei einer relationalen Datenbank von nicht sehr großer Bedeutung. Die meisten Befehle benötigen keine sortierten Dateien für ihre Ausführung.

Weitaus wichtiger ist die Indizierung von Dateien in dBase II. Wenn man sehr große Dateien sortiert, ist das eine sehr zeitaufwendige Sache. Arbeitet man mit indizierten Dateien, so wird ein neuer Datensatz schon während der Eingabe einsortiert. Mit der Möglichkeit, auf indizierten Dateien zu arbeiten, kann das Sortieren ganz erheblich beschleunigt werden. Oder wenn nach Sätzen gesucht werden soll und dafür zwei oder mehr Kriterien in Frage kommen, wird dieser Vorgang durch Verwendung einer indizierten Datei wesentlich schneller.

Die Indexdatei ist eine Datei, die nach einem oder mehreren Schlüsseln sortiert ist. Sie enthält pro Satz nur noch die im Indexbefehl angegebenen Felder. Dieser sogenannte Zeiger (Pointer) verweist auf den zugehörigen Satz in der Stammdatei. Damit steht für eine Bearbeitung der vollständige Inhalt der Ursprungsdatei zur Verfügung. Eine aktuelle Datei wird indiziert durch:
INDEX ON <Feldname> TO <Indexdatei>

Die so erzeugte Indexdatei wird unter dem Namen <Indexdatei> abgelegt. »Feldname« kann hier auch eine Aufzählung von Feldern sein. Dieser erweiterte Index-Befehl könnte für das Beispiel in Bild 12 dann so aussehen:
INDEX ON name, vorname, postltzahl TO bspdatei

Die Datei namens »bspdatei« wäre nach Befehlsausführung nach drei Schlüsseln sortiert (nämlich »name«, »vorname« und »postltzahl«).

Für sehr schnelle Suchvorgänge in Indexdateien ist der Befehl
FIND <Suchbegriff>
geeignet, der vor allem für große Dateien nützlich ist. Da pro Datensatz nur das indizierte Feld abgefragt wird, ist die Suche mit FIND außerordentlich schnell. Voraussetzung für eine solche Suche ist natürlich, daß die Indexdatei nach dem Suchbegriff indiziert ist.

Programmieren mit dBase II

Durch den interaktiven Frage-Befehl (interaktiv nennt man die Beziehungen zwischen Mensch und Computer, das heißt, der Mensch fragt und der Computer antwortet direkt)
? <Ausdruck>
kann ein Feldinhalt ausgegeben oder gedruckt werden. Darüber hinaus kann man mit dem Kommando
? <mathematische Formel>
in dBase auch rechnen. »?« entspricht in Basic dem PRINT-Befehl.

Für eine präzise Positionierung von Texten und Daten ist das Kommando
@ <Zeile>,<Pos.> SAY <Ausdruck>
gedacht.

Das Zeichen »@« wird auch AT-Zeichen oder Klammeraffe genannt. Der Klammeraffen-Befehl entspricht in Basic dem PRINTAT-Befehl. Er erfüllt eine sehr wichtige Funktion beim Formatieren und Beschreiben von Druckformularen. Ebenso wichtig ist er beim Aufbau von Bildschirmmasken. Diese Masken wurden schon anhand von Bild 9 beschrieben. Im Bild 13 sieht man ein Programm-Listing, das die Positionierungsbefehle der Bildschirmmaske (Bild 14) enthält. Nur ganz kurz zur Bedeutung einzelner Befehle: Der Klammeraffen-Befehl in der ersten Zeile besagt
»drucke in der ersten Zeile an der 30. Stelle den Text 'Programm-Auswahl'«.

In der 5. Zeile an der 5. Stelle beginnt der Vorname (»TRIM« = ohne Leerstellen). Direkt dahinter (+ = verknüpfe mehrere Zeichenketten) wird ein Leerzeichen (' ') und dann der Nachname gedruckt.

Die Bildschirmmaske in Bild 14 wird durch dieses Programm aufgebaut. Man sieht: Vorname und Nachname stehen – durch ein Leerzeichen getrennt – in der 5. Zeile an der richtigen Stelle. Hinter den Angaben »bisheriger Umsatz« und »offene Posten« sind Bildschirmfelder vorgesehen, die hier leider nicht sichtbar werden. Hinter »DM« kann der jeweilige Betrag einfach eingegeben werden.

In dBase II sind vier grundlegende Programmstrukturen verfügbar:
- eine Abfolge von Befehlen
- eine Auswahl von Entscheidungen
- Wiederholung (Schleifen)
- Prozeduren (Unterprogramme)

Alle Kommandos und Befehle können direkt eingegeben und schrittweise ausgeführt werden. Wenn der Anwender anstelle von Direktbefehlen über Befehlsfolgen in Programmform verfügen will, stellt dBase dafür Kommandos zur Verfügung. Um ein Programm zu schreiben, das immer wieder ausgeführt werden kann, wird zuerst durch das Kommando
MODIFY COMMAND <Programm-Name>
eine Datei angelegt, die das Listing des Programms enthalten soll. Das in die neue Kommandodatei eingegebene Programm wird aufgerufen mit
DO <Programm-Name>

Mit dem MODIFY-Kommando, das eine Programmdatei anlegt, kann eine bereits bestehende Kommandodatei wieder editiert, geändert und erweitert werden.

Beispiele:
- »WHILE«-Schleife

Will man einen wiederholten Zugriff auf Daten realisieren, so verwendet man Programmschleifen, die beliebig oft durchlaufen werden können.

Das folgende Mini-Beispielprogramm bearbeitet eine Datei Satz für Satz, bis das Dateiende erreicht ist.



```
DO WHILE .NOT. EOF
    <Befehlsfolge>
    SKIP
ENDDO
```

Durch den Befehl SKIP (=springen) wird der jeweils folgende Datensatz aufgerufen. Das Prädikat »EOF« (=END OF FILE = Dateiende) prüft, ob das Dateiende der geöffneten Datei schon erreicht ist. Wenn ja, dann liefert EOF den logischen Wert »T« (= wahr), sonst »F« (= falsch). Die obige »WHILE«-Schleife wird durchlaufen, solange der Wert von »EOF« nicht wahr ist. Das heißt, solange das Dateiende noch nicht erreicht ist, kann die Datei weiter bearbeitet werden.

– »CASE« – mehrfache Alternativen

In Bild 15 wird gezeigt, wie auf Benutzereingaben fallweise unterschiedlich reagiert werden kann. Wie das funktioniert, wurde im Abschnitt »Anwendungsmöglichkeiten für dBase II« schon beschrieben.

Mit Hilfe der »IF«-Auswahl kann zwischen zwei Möglichkeiten unterschieden werden. Das kleine Beispiel zeigt folgendes: Wenn das Dateiende – das heißt »EOF« = T – erreicht ist, wird eine entsprechende Meldung gedruckt. Im »ELSE«-Teil weiß man, daß noch unbearbeitete Sätze der Datei vorliegen, es wird eine beliebige Befehlsfolge ausgeführt, die die Daten verarbeitet.

```
IF EOF
  @ 5,5 SAY "Datei-Ende
  erreicht"
ELSE
<bearbeite den nächsten
Datensatz>
ENDIF
```

Die &-Makrofunktion:

Die wiederholte Eingabe von Zeichenketten läßt sich in dBase II einfach vermeiden, indem man sie einmal einer Variablen zuweist. Nennen wir diese Variable »T«. Diese Variable tritt im Programm dann mit ihrem Namen stellvertretend (als Makro) für die Zeichenkette auf. Erst wenn die Zeichenkette benötigt wird, erfolgt eine Ersetzung. Ersetzt wird der Name der Variablen, der nun mit dem Zeichen »&« beginnt. Er heißt nun »&T«. Dieser Ersetzungsvorgang wird automatisch durch das Programm durchgeführt. In Bild 6 sieht man, wie solche Variablen sinnvoll eingesetzt werden können.

Beispiel:

```
.STORE "DELETE RECORD" TO T
DELETE RECORD
.&T 5
```

Die Variable T wird im ersten Kommando auf den Wert »DELETE RECORD« gesetzt. dBase II meldet sich daraufhin und listet den gespeicherten Text noch einmal.

In der dritten Zeile wird dann der Variablenname »&T« vom Programm durch den Text »DELETE RECORD« ersetzt. Nun wird aus dem Variablenwert und der Zahl 5 der Befehl DELETE RECORD 5 zusammengesetzt und kann ausgeführt werden.

Mit dieser Makrofunktion lassen sich auch Parameterwerte zwischen Kommando-Dateien austauschen, wenn diese sich gegenseitig aufrufen.

Anmerkungen zur Peripherie

Floppies:

Bei der Anwendung von dBase II bekommt man, wie generell bei CP/M-Software auf dem C 128, Platzprobleme, wenn man mit der Floppy 1541 arbeitet. Die Floppy ist nicht nur um den Faktor 10 langsamer als die 1571/70, was sich beim Arbeiten mit dBase II ganz besonders bemerkbar macht. Ernste Speicherplatzprobleme gibt es, wenn man nur ein einziges Laufwerk besitzt. CP/M und dBase II belegen zusammen soviel Platz, daß kein Raum mehr da ist, um Dateien anzulegen. Auch ein Diskettenwechsel ist ausgeschlossen, da das Betriebssystem CP/M immer wieder Programme nachlädt. Aber auch mit den Floppies 1571/70 kann man optimal nur mit zwei Laufwerken arbeiten. Von den 340 kByte, die man auf einer 1571 zur Verfügung hat, sind etwa 200 kByte für die Daten frei (CP/M belegt 59 kByte, die dBase-Programm-Files 124 kByte). Zum Vergleich: Eine dBase II-Datei mit 28 Sätzen mit jeweils insgesamt 150 Zeichen belegt schon 6 kByte.

Drucker:

Das Programm »SETUP« wird ausgeführt, bevor dBase II gestartet wird. Mit »SETUP« kann zwischen DIN- oder ASCII-Tastatur gewählt werden. Angeschlossen werden können serielle Drucker, parallele Centronics-Drucker über den User-Port und Centronics-Drucker mit VC-Interface, die über den seriellen Bus betrieben werden. Über den seriellen Bus können auch die Commodore-Drucker angeschlossen werden. Da Commodore-Drucker nicht über den deutschen Zeichensatz verfügen, sind sie zum Arbeiten mit der vorliegenden deutschen Version von dBase II nur dann zu empfehlen, wenn man auf Umlaute verzichten kann. Keine Probleme machen dagegen parallele Drucker mit oder ohne VC-Interface.

Fazit:

Wenn man über die geeignete Peripherie verfügt, ist dBase II das beste Datenbanksystem, das zur Zeit für PCs verfügbar ist. Bevor man dBase II einsetzt, muß auf jeden Fall die Platzfrage geklärt sein! Ohne genug Diskettenplatz für die Daten kann dBase II nicht sinnvoll eingesetzt werden. Das Sortieren in dBase II ist schon für kleine Dateien sehr langsam. Dieser Nachteil wird relativ gut ausgeglichen, wenn man mit indizierten Dateien arbeitet.

Die Beispiele stammen zum Teil aus der Einführung von Dr. Peter Albrecht, dBase II für den Commodore 128 PC. Die Tabelle 1 ist entnommen aus dem Handbuch zum Programm. (cg)

Literatur:

Dr. Peter Albrecht, dBase II für den Commodore 128 PC, Markt&Technik Verlag 1985, ISBN 3-89090-189-1, 49 Mark

Dr. Peter Albrecht, Das Datenbanksystem dBase II, Markt&Technik Verlag 1985, ISBN 3-89090-143-3, 291 Seiten, 68 Mark

Wayne Ratliff, dBase II Anwender-Handbuch, Ashton-Tate GmbH, Markt&Technik Software-Verlag 1984, 437 Seiten, 49 Mark

maximal 65535 Sätze pro Datei-File,
maximal 1000 Zeichen pro Satz,
maximal 32 Felder pro Satz,
maximal 254 temporäre Variablen,
größte darstellbare Zahl:
etwa +1,8 x 10 hoch 63,
kleinste darstellbare Zahl:
etwa +1 x 10 hoch -63,
Rechengenauigkeit: 10 Stellen,
maximal 254 Zeichen pro Zeichenkette,
maximal 254 Zeichen pro Befehlszeile,
maximal 99 Zeichen Index-Schlüssel-Länge,
maximal 5 Ausdrucke in einem SUM-Befehl.
Notwendiges System: C 128
Floppy: 1571/70, besser mit 2 Laufwerken
Preis: 199 Mark

Tabelle 1. Die wichtigsten Daten von dBase II

Befehle von dBase II nach Funktionsgruppen geordnet

Befehle zur Bearbeitung der Datei-Struktur

CREATE
Erzeugt eine neu zu strukturierende Datenbank.

CREATE <neue datei> FROM <alte datei>
Erzeugt die <neue datei> automatisch mit der Datenstruktur, die in den Datensätzen der <alten datei> beschrieben wird. (Siehe auch COPY STRUCKTURE EXTENDED).

USE <alte datei>
COPY [bereich] TO <dateiname> [STRUCTURE] [FIELD <liste>] [FOR <ausdr>]
Kopiert Daten aus einer Datenbank in eine andere Datei.

USE <alte datei>
COPY TO <dateiname> STRUCTURE EXTENDED
Erzeugt eine neue Datei, deren Datensätze die Struktur der alten Datei beschreiben (siehe auch CREATE <neue datei> FROM <alte datei>).

CREATE <neue datei> FROM <alte datei>
DISPLAY STRUCTURE
LIST STRUCTURE
Beide Anweisungen zeigen die Struktur der gerade bearbeiteten Datei.

MODIFY STRUCTURE
Ändert die Struktur der Datensätze in einer Datenbank (Feldnamen, Größen und Gesamtstruktur). Löscht alle Daten in der Datenbank.

Veränderung einer Datenbankstruktur mit Daten in der Datenbank:

USE <alte datei>
COPY TO <neue datei>
USE <neue datei>
MODIFY STRUCTURE
APPEND FROM <alte datei>
COPY TO <alte datei>
USE <alte datei>
DELETE FILE <neue datei>

Umbenennen von Datenfeldern mit Daten in der Datenbank:

USE <alte datei>
COPY TO <neue datei> SDF
MODIFY STRUCTURE
APPEND FROM <neue Datei>.TXT SDF
DELETE FILE <neue Datei>

Befehle zur Dateibearbeitung:

USE <dateiname>
eröffnet eine Datei und nimmt sie in Bearbeitung.

USE <neue datei>
schließt die alte Datei.

USE
schließt alle Dateien.

RENAME <alter Name> TO <neuer Name>
Ändert den Namen einer Datei. NICHT auf die gerade bearbeitete Datei anwenden!

COPY TO <dateiname>
erzeugt eine (identische) Sicherheitskopie.

CLEAR
schließt alle Dateien und löscht alle Speicher-Variablen.

SELECT [PRIMARY] [SECONDARY]
Erlaubt zwei Dateien zur gleichen Zeit unabhängig voneinander eröffnet zu sein und abwechselnd bearbeitet zu werden (Vorder- und Hintergrund-Arbeitsfeld). Daten können zwischen den Dateien mit dem Namen-Präfix "P." bzw. "S." übertragen werden.

DISPLAY FILES [ON <|>]
listet die Datenbanken, die auf dem eingerasteten ("I" = logged in) oder dem angegebenen Laufwerk gespeichert sind. Statt dessen kann auch der Befehl LIST benutzt werden.

DISPLAY FILES LIKE <jokerzeichen> [ON<|>]
zeigt andere Arten von Dateien auf den Laufwerken.

DELETE FILE <dateiname>
löscht eine Datei auf der Diskette.

QUIT
schließt beide Arbeitsfelder (Vorder- und Hintergrund), alle Dateien und beendet die Arbeit mit dem dBASE II-System.

Sortier-Befehle

SORT ON <schlüssel> TO <neue datei>
Sortieren der Datensätze in einer Datei nach einem Schlüssel. Der Befehl verändert die Reihenfolge der Sätze in der sortierten Datei.

INDEX ON <schlüssel> TO <neue datei>
Indizieren von Datensätzen in einer Datei (Erzeugen eines Index-Files), auch mit mehrfachen Schlüsseln. Der Befehl verändert die Satznummer nicht.

Befehle zur Bearbeitung zweier Datenbanken

COPY TO <neue datei>
erzeugt ein Duplikat der gerade bearbeiteten Datei.

APPEND FROM <andere datei>
fügt Datensätze zu der in Bearbeitung befindlichen Datei.

UPDATE FROM <andere datei> ON <schlüssel>
summiert zu Übersichten oder ersetzt die Daten in der in Bearbeitung genommenen Datei aus der anderen Datei. Beide Dateien müssen nach dem Schlüssel sortiert sein.

JOIN
erzeugt aus zwei Dateien eine dritte Datei.

Befehle zum Editieren, Aktualisieren, Verändern von Daten

DISPLAY,LIST,BROWSE
zeigen den Inhalt von Datensätzen an.

DELETE
markiert einen Datensatz, der gelöscht werden soll.

RECALL
macht die Lösch-Markierung (DELETE) rückgängig.

PACK
entfernt Datensätze mit Lösch-Markierung (DELETE) endgültig aus der Datenbank.

EDIT
erlaubt Verändern der Einträge bestimmter Datensätze.

REPLACE <feld WITH daten>
globales Ersetzen von Daten in bestimmten Feldern, kann mit Bedingungen erweitert werden, wie die meisten dBASE II-Befehle.

CHANGE FIELD
Editierung auf der Grundlage von Feldern statt Datensätzen.

@ <koord> [SAY['nachricht']] [GET<var>[PICTURE]]
READ stellt die Variablen dar und erlaubt die Eingabe neuer Werte.

INSERT [BEFORE] [BLANK]
fügt einen Datensatz in eine Datenbank ein.

UPDATE FROM <andere datei> ON <schlüssel>
summiert zu Übersichten oder ersetzt die Daten in der in Bearbeitung genommenen Datei aus der anderen Datei. Beide Dateien müssen nach dem Schlüssel sortiert sein.

MODIFY COMMAND <dateiname>
erlaubt das Bearbeiten Ihrer Programmdatei, ohne daß Sie dazu Ihren Texteditor aus dem Betriebssystem aufrufen müssen.

Befehle zum Gebrauch von Variablen

Es können bis zu 64 Speicher-Variable und eine beliebige Zahl von Feldnamen benutzt werden).

LIST MEMORY, DISPLAY MEMORY
beide Anweisungen geben die erzeugten Variablen, Ihre Datentypen und ihren Inhalt aus.

& ersetzt den Inhalt einer Variablen vom Typ-Zeichen durch ihren Inhalt (das heißt, erzeugt den Effekt, als stünde anstelle des Variablen-Namens der Wortlaut des Textes, den die Variable speichert (Makro-Aufrufe etc.).

STORE <wert> TO <var>
erzeugt Variable oder ändert Ihren Inhalt.

RELEASE <var>
löscht die genannte Variable.

SAVE MEMORY TO <dateiname>
speichert Variable in der angegebenen Datei (mit der Namen-Erweiterung.MEM).

RESTORE FROM <dateiname>
liest Variable in den Speicher zurück (und löscht dabei zuvor vorhandene Speichervariable.)

Befehle für interaktive Eingabe (Dialoge):

WAIT
hält Programmausführung, Scrollen des Bildschirminhaltes etc. an, bis irgendeine Taste gedrückt wird.

WAIT TO <var>
speichert das Zeichen in einer Speichervariablen.

INPUT ['nachricht'] TO <var>
akzeptiert alle Datentypen und speichert sie in einer Variablen (die erzeugt wird, falls sie

Tabelle 2. Tabelle der dBase II-Befehle

noch nicht existiert). Zeichen-Eingaben müssen in Anführungszeichen eingeschlossen werden.

ACCEPT ['nachricht'] TO <var>
wie INPUT, aber ohne Anführungszeichen bei textueller Eingabe (Zeichenketten).

@ <koord> SAY ['nachricht'] GET <var> [PICTURE] READ
stellt Variable auf dem Bildschirm dar und liest neue Werte in sie ein.

Suchbefehle

SKIP [+ <ausdr>]
springt um eine bestimmte Anzahl von Datensätzen vor- oder rückwärts.

GO [TO] <zahl>, **GO TOP, GO BOTTOM**
bringt Sie zu einem bestimmten Datensatz, dem ersten Datensatz (GO TOP), oder zum letzten Datensatz (GO BOTTOM) in der Datei.

FIND <zeichkette>
sucht einen Datensatz in einer indizierten Datenbank nach dem Wert des Schlüssels.

LOCATE FOR <ausdr>
CONTINUE
Findet den Datensatz, der mit der Bedingung <ausdr> übereinstimmt.

Ausgabe-Befehle

?,DISPLAY, LIST
zeigen (Werte von) Ausdrücken, Datensätzen, Variable, Strukturen.

REPORT [FORM <formatname>]
erzeugt Datenbankauswertungen.

@ <koord> **SAY** <var/ausdr/zeichk>
formatiert Ausgaben auf dem Bildschirm oder dem Drucker.
[USING <format>] kann hinzugefügt werden, um das PICTURE-Format auf dem Drucker zu erzeugen.

Programm-Befehle für Befehls-Dateien:
(dBASE II-Programme werden in sogenannten »command files« (Befehls-Dateien) gespeichert, deren Namen die Erweiterung ».CMD« beziehungsweise »PRG« besitzen).

DO <dateiname>
startet die Ausführung eines Programms.

IF <bedingungen>
auszuführende Befehle
ELSE
alternativ auszuführende Befehle
ENDIF
Trifft einfache Entscheidungen (ein oder zwei Wege) oder mehrfache, wenn verschachtelt.

DO CASE CASE <ausdr>
auszuführende Befehle ... OTHERWISE
ENDCASE
Ähnlich wie IF ENDIF, aber ohne Verschachtelung.

DO WHILE <bedingungen> [LOOP]
zu wiederholende Anweisungen
<Abbruchkriterium>
ENDO
Schleifen-Anweisung. Die »Bedingungen« müssen durch irgendeine Anweisung in der Schleife verändert werden, damit die Schleife nicht »ewig« läuft.

Turbo Pascal auf dem C 128

Turbo-Pascal gilt als einer der schnellsten und leistungsfähigsten und mit 198 Mark einer der preisgünstigsten Pascal-Compiler überhaupt. Unter CP/M ist Turbo-Pascal nun auch für den C 128 verfügbar.

Wer einen C 128 besitzt, der wird sich sicherlich auch schon mit dem CP/M-Modus seines Computers beschäftigt haben. Denn mit CP/M eröffnet sich eine der größten Softwaresammlungen überhaupt. So wird auch das vielgerühmte Turbo-Pascal endlich für den Commodore-Anwender verfügbar.

Was bietet die moderne und weit verbreitete Sprache Pascal dem C 128-Anwender überhaupt? Hier ein kurzer Einblick:

Mit Pascal lernt man eine strukturierte Sprache, bei der man den mitunter recht chaotischen Programmierstil von Basic sehr schnell vergißt. Diese nach dem französischen Mathematiker Blaise Pascal benannte Sprache fordert eine genaue Erfassung des Problems, das programmiert werden soll. Mit einfachem Drauflosprogrammieren wird man daher nicht sehr weit kommen.

Ein besonderes Merkmal ist die Tatsache, daß unübersichtliche GOSUB- und GOTO-Anweisungen mit Zeilennummern wegfallen und durch höhere Sprachstrukturen ersetzt werden.

Pascal ermöglicht eine strukturierte Aufgliederung eines Programms in logische Module (Prozeduren), die alle bestimmte Teilaufgaben des Problems lösen. Alle benutzten Variablen in einem Modul sind, wenn nicht vorher anders definiert, lokal, das heißt, nur in diesem einen Unterprogramm gültig und veränderbar. Innerhalb der einzelnen Prozeduren wie auch im Hauptprogramm herrschen strenge Syntax- und Aufbauregeln, die für logische Struktur und gute Übersicht sorgen.

Nebenbei sei erwähnt, daß Pascal als Compilersprache selbstverständlich sehr viel schneller arbeitet als das interpretative Basic.

Was bringt Turbo-Pascal?

Turbo-Pascal ist stark dem Standard-Pascal von Nikolaus Wirth, dem Schöpfer von Pascal, angepaßt, hat aber noch einige Zusätze und Erweiterungen parat, die das Herz eines jeden Pascal-Fans höher schlagen lassen.

Mit ABSOLUTE kann der Programmierer die normalerweise vom Compiler vergebene absolute Speicheradresse von Variablen festlegen. EXTERNAL dient zum Aufruf von in Assembler geschriebenen und getrennt übersetzten Prozeduren und Funktionen. Mit INLINE kann Maschinensprache direkt in ein Pascal-Programm integriert werden, was bei zeitkritischen Anwendungen von Bedeutung sein kann. Die sehr maschinennahen Operationen SHL und SHR entsprechen den Schiebeoperationen in Assembler.

Das Attribut PACKED wird vom Turbo-Pascal Compiler zwar ohne Murren akzeptiert, hat aber keine Wirkung, da Turbo-Pascal ohnehin schon einen sehr komprimierten Code erzeugt.

Doch damit ist das »Sonderzubehör« noch nicht erschöpft. Turbo-Pascal stellt noch eine Reihe von Standard-Prozeduren, -Funktionen und -Konstanten bereit, die die Programmierung erleichtern und die Möglichkeiten der Problemlösung erweitern.

Die Kreiszahl Pi ist dem Turbo Compiler neben anderen spezifischen Konstanten auch ohne vorherige Definition auf elf Stellen genau bekannt.

Ebenfalls implementiert sind viele Funktionen zur Zahlen- und Variablenverarbeitung. So ist für Turbo-Pascal das Bestimmen einer Zufallszahl mit RANDOMIZE und

die komfortable Behandlung von Dateien eine Kleinigkeit.

Der wichtige und oft gebrauchte Datentyp STRING, den man sich in anderen Pascal-Versionen erst selbst definieren muß, ist in Turbo-Pascal bereits standardmäßig vorhanden. So können Strings ganz zwanglos deklariert werden:
VAR EINGABE: STRING(80);
NAME: STRING(16);

Derart verschieden dimensionierte Strings sind untereinander dennoch kompatibel, das heißt, sie können untereinander zugewiesen werden, wobei natürlich der eventuell längere Teil des zugewiesenen Strings abgeschnitten wird.

Overlay-Technik löst Speicherplatzprobleme

Wenn ein größeres Pascal-Projekt nicht ganz in den begrenzten Arbeitsspeicher des Computers paßt, dann mag das für andere Programmiersprachen ein ernstes Problem darstellen, nicht so jedoch für Turbo-Pascal. Durch das Attribut OVERLAY können Pascal-Prozeduren oder Funktionen als Overlay-Files compiliert werden. Im Speicher wird dann lediglich Platz für die längste aller Overlay-Prozeduren reserviert. Beim Aufruf einer solchen Prozedur oder Funktion wird das entsprechende Overlay-File nachgeladen und ausgeführt. Insbesondere bei Prozeduren, die nur wenige Male im Verlauf des Programms aufgerufen werden, ist diese Overlay-Technik sehr effektiv.

Eine bemerkenswerte Anzahl von Prozeduren ist zur direkten Manipulation des Speichers vorgesehen. Es lassen sich zum Beispiel ohne Schwierigkeiten Maschinenprogramme oder andere Programmdateien vom Pascalprogramm aus laden und starten. Mit einigen Prozeduren können sogar Routinen aus dem BIOS (Basic Input/Output System) und dem BDOS (Basic Disc Operating System) von CP/M angesprungen und somit das CP/M-Betriebssystem voll ausgenutzt werden.

Mit den gleichnamigen Prozeduren BIOS, BIOSHL, BDOS und BDOSHL lassen sich gezielt Routinen des CP/M-Systems aufrufen und für Pascal-Programme nutzbar machen; für CP/M-Kenner geradezu ein Leckerbissen.

Da Turbo-Pascal unter CP/M läuft, ist es notwendig, zunächst CP/M von der Systemdiskette in Ihren C 128 zu laden. Wie Sie vielleicht bereits bemerkt haben, tut dies Ihr C 128 von selbst, wenn Sie ihn anschalten und die Systemdiskette im Diskettenlaufwerk steckt.

Turbo-Pascal und der C 128

Er »bootet« (lädt und startet) CP/M automatisch. Das ist sowohl mit der 1570/1571-Floppy als auch mit der guten alten 1541 möglich. Von der Verwendung des 1541-Laufwerks kann allerdings nur abgeraten werden. Allein das Hochfahren des CP/M-Systems benötigt fast zwei volle Minuten. Meldet sich nun endlich CP/M auf dem Bildschirm, darf die Diskette mit Turbo-Pascal in das Floppylaufwerk geschoben werden. Wenn Sie noch keine Kopie der Original-Diskette gemacht haben, dann sollten Sie das spätestens an dieser Stelle tun. Wie alle CP/M-Software wird auch Turbo-Pascal ohne unnütze Kopierschutz-Pfuschereien geliefert. Dafür hat jede einzelne Diskette eine Seriennummer. Wenn man sich nach dem Kauf mit einer beigefügten Postkarte unter Angabe der Seriennummer als Benutzer registrieren läßt, hat man die beruhigende Gewißheit, bei Problemen mit dem Produkt nicht allein gelassen zu werden. Bei anderen bekannten Software-Produkten erschöpft sich die »Unterstützung« des Benutzers ja oftmals mit dem Hinweis »Telefonische Anfragen können leider nicht beantwortet werden«.

Nachdem man also jetzt die Originaldiskette sicherheitshalber kopiert hat, kann's endgültig losgehen. Mit DIR listen Sie das Inhaltsverzeichnis der Diskette und können mehrere Files darauf finden.

TURBO.COM ist der eigentliche Compiler, der die gesamte Arbeit leistet. Er benötigt zirka 32 KBytes Speicher. TURBO.OVR wird beim Arbeiten mit Overlay-Prozeduren benötigt. TURBO.MSG enthält alle Compiler-Meldungen im Klartext und kann auf Wunsch von Turbo-Pascal aus geladen werden.

TLIST.COM kann, wenn ein Drucker vorhanden, den Quelltext Ihrer Pascal-Programme auf dem Drucker ausgeben. Ein weiteres File mit dem Namen READ.ME gibt nach dem Auflisten mittels TYPE-Kommando Informationen über die Implementation von Turbo-Pascal.

Daneben gibt es noch einige Beispielprogramme, mit denen man herumexperimentieren kann. Dazu gehört auch ein hervorragendes Tabellenkalkulationsprogramm namens MC, das vollkommen in Turbo-Pascal geschrieben ist.

Turbo Pascal wird einfach durch Eingabe von TURBO gestartet. Nach wenigen (1571) oder vielen (1541) Sekunden schließlich meldet sich Turbo-Pascal und erkundigt sich, ob ausführliche Pascal-Fehlermeldungen erwünscht sind. Antwortet man mit Ja, so wird das File TURBO.MSG, welches etwa 1400 Bytes lang ist, nachgeladen. Wird das unterlassen, dann gibt der Compiler bei etwaigen Fehlermeldungen nur deren Nummer aus.

Es dauert nun nicht mehr lange, bis Turbo-Pascal endgültig sein Hauptmenü (Bild 1) zeigt und dem Programmieren nichts mehr im Wege steht.

Das Arbeiten mit Turbo-Pascal

Turbo-Pascal ist ein Resident-Compiler, das heißt, ist er einmal im Speicher, benötigt er für seine Arbeit keine weiteren Diskettenzugriffe mehr und bleibt solange aktiv, bis Sie Ihren C 128 ausschalten oder in die CP/M-Ebene zurückkehren.

Turbo-Pascal ist sehr bedienerfreundlich und gibt auch dem Anfänger wenig Rätsel auf, so daß man die richtigen Handgriffe sehr schnell beherrschen lernt. Aus dem eben erwähnten Menü kann man alle wichtigen Funktionen auf Tastendruck abrufen.

Der für den Programmierer wichtigste Programmteil dürfte wohl der Eingabeeditor sein, der mit »E« gestartet wird. Haben Sie noch kein Pascal-Quellprogramm im Speicher, muß jetzt zunächst der Name des Files eingegeben werden, unter dem es nachher abgespeichert werden kann.

Narrensicher wie Turbo-Pascal ist, sieht es sofort nach, ob ein derartiges File schon einmal auf die Diskette geschrieben wurde. Falls das so ist, wird es eingelesen, und man kann dieses File im Editor bearbeiten. Wenn nicht, richtet der Compiler ein neues File ein. Jedes Programm oder File in CP/M besitzt eine Art Kennung am Ende des Filenamens, die durch einen Punkt getrennt so etwas wie eine Identifikation darstellt. Turbo-Pascal hängt, wenn nur der Dateiname eingegeben wird, das Kürzel PAS an, um dem Benutzer zu kennzeichnen, daß es sich bei diesem File um eine Turbo-Pascal Quelldatei handelt. Will man das nicht, kann man eine eigene Kennung benutzen.

```
Logged drive: A

Work file:
Main file:

Edit     Compile    Run     Save
Quit        compiler        Options

Text: 0       Bytes
Free: 62903   Bytes

>■
```

Bild 1. Das Hauptmenü von Turbo-Pascal

Professioneller Texteditor

Wem der Name »Wordstar« ein Begriff ist und wer eventuell schon damit gearbeitet hat, der darf sich freuen. Die Kommandos des Pascal-Editors sind mit den entsprechenden Wordstar-Befehlen identisch und haben, wenn nicht anders vereinbart, auch die gleiche Tastenbelegung. Wer das Wordstar-Textverarbeitungssystem noch nicht kennt, wird sich dann wohl erst einmal mit dem recht ausführlichen Turbo-Pascal Handbuch, Stichwort Editor, vertraut machen müssen.

Doch obwohl die Fülle der Kommandos auf den ersten Blick überwältigend ist (selbst Block-, Such- und Ersetz-Kommandos sind vorhanden), gestaltet sich das Arbeiten mit dem Turbo-Pascal-Editor doch sehr einfach und angenehm.

Der Cursor wird neben den Wordstar-üblichen Control-Kommandos auch mit den vier Cursortasten bewegt, die sich gleich links neben den Funktionstasten des C128 befinden. Auch die INST/DEL Taste hat ihre Funktion behalten, so daß man also auch ohne nähere Kenntnisse der einzelnen Editor-Kommandos gleich mit dem Programmieren beginnen kann.

Interessant dürfte sein, daß beim Druck der Return-Taste der Cursor nicht an den ganz linken Rand der neuen Zeile springt. Stattdessen plaziert er sich immer unter dem ersten Wort der zuletzt eingegebenen Zeile. Das Programmieren mit den für Pascal so wichtigen Einrückungen von Zeilen wird durch diese kleine Feinheit zum Kinderspiel.

Es lassen sich bis zu 126 Zeichen in einer Zeile unterbringen, wobei der Editor den Bildschirm bei der Eingabe von mehr als 80 Zeichen seitwärts scrollen läßt.

Hat man sein Programm eingegeben, kann man den Editor (mit Control KD) wieder verlassen und das frisch geschriebene Programm sofort übersetzen und ausprobieren.

Über seinen Quelltext muß man sich dabei nicht groß sorgen: Turbo-Pascal geht stets auf Nummer sicher und fragt immer dann, wenn die Gefahr besteht, den momentan im Speicher befindlichen Quelltext zu verlieren, ob das aktuelle Quellprogramm abgespeichert werden soll. Sei es beim Verlassen des Compilers, oder wenn man ein neues Programm bearbeiten will, Turbo-Pascal erinnert den Anwender an das Sichern des Quelltextes, es sei denn, man hätte dies bereits getan.

Der Turbo Compiler

Der Turbo Compiler besitzt mehrere Funktionen zum Compilieren eines Programms. Mit dem O-Kommando des Hauptmenüs läßt sich ein Untermenü mit den möglichen Compiler-Optionen aufrufen, die das Übersetzen von Quelltexten auf verschiedene Art ermöglichen. Je nachdem, welche Option man wählt, wird der Compiler bei Aufruf entsprechend arbeiten.

Turbo-Pascal erlaubt es auf Wunsch, das im Moment im Speicher befindliche Quellprogramm zu compilieren und auch direkt wieder im Speicher abzulegen. Das so erzeugte Maschinensprache-Programm wird dann vom Hauptmenü aus mit »R« gestartet. So kann ein sofortiger Test des eben geschriebenen Programms erfolgen.

Der Compiler kann aber auch ein Quellfile von Diskette laden und compilieren, um das Compilat (das übersetzte Programm) dann als .COM-File wieder auf Diskette zu speichern. Dieses File ist später als Programmfile von CP/M aus, ohne Hilfe von Turbo-Pascal, lade- und startfähig, somit also ein vollkommen eigenständiges Programm.

Schließlich kann der Compiler auch sogenannte Chain-Dateien (Kennung .CHN) verarbeiten. Diese Übersetzungen haben keine eigenen Pascal-Bibliotheksroutinen und sind somit auch nicht einzeln für sich lauffähig. Sie sind dafür gedacht, von anderen Pascal-Programmen nachgeladen zu werden.

Der letzte Untermenüpunkt mit der Bezeichnung »Find run-time error« ist genau das, was sein Name verspricht: ein gutes Hilfsmittel zum Aufspüren von Laufzeitfehlern. Laufzeitfehler sind Fehler, die der Compiler nicht erkennen kann, da sie erst beim Ablauf des compilierten Programms auftreten. In diesem Falle unterbricht das Programm und gibt eine entsprechende Fehlermeldung und die Speicheradresse aus, an der der Fehler aufgetreten ist.

Nun besteht ein Compilat nicht mehr aus den Pascalbefehlen, die man im Quelltext eingetippt hat, sondern aus Maschinencode. Die angezeigte Stelle eines Fehlers ist deshalb auch eine hexadezimale Adresse. Dank der eben angesprochenen Option des Turbo-Pascal Compilers kann man aber mit dieser Adresse den Lauffehler im Quellprogramm jedoch bequem wiederfinden. Dazu muß das Quellprogramm noch einmal von Turbo-Pascal geladen werden. Ein Aufruf der oben genannten Find-Option und die Angabe der besagten Hexadresse genügen, und der Editor führt Sie automatisch an die Fehlerquelle. Eine praktische Sache.

Doch bevor ein Laufzeitfehler auftreten kann (man will es nicht hoffen), muß man sein Quellprogramm selbstverständlich erst compilieren lassen.

Mit der Wahl »C« im Hauptmenü startet der Compiler und compiliert je nach eingestellter Compiler-Option aus eben besprochenem Untermenü. Die Fehlerbehandlung während der Übersetzung geschieht bei Turbo-Pascal auf sehr interessante und effektive Weise. Hat der Compiler einen Fehler gefunden, den er nicht akzeptieren kann, läßt er eine entsprechende Fehlermeldung auf dem Bildschirm erscheinen. Falls man die Fehlermeldungen beim Start von Turbo-Pascal nicht geladen hat, wird nur die Fehlernummer ausgegeben, deren Bedeutung im Handbuch nachzulesen ist. Der Übersetzungsvorgang wird gestoppt und ein Druck auf die Escape-Taste (ganz links oben auf der Tastatur) wird erwartet.

Folgt man dieser Aufforderung, startet der Editor automatisch und postiert den Cursor ähnlich wie bei

der Laufzeitfehlerbehandlung an die fehlerhafte Programmstelle. Eine überaus effektive Fehlersuche ist damit gewährleistet, die man bald nicht mehr missen möchte.

Schnell, schneller, turbo...

Eine Auflistung des Programmes während der Compilation ist nicht möglich. Sie würde die Übersetzung auch nur unnötig verlangsamen.

Der Name »TURBO« hält, was er beim Lesen verspricht. Turbo-Pascal erledigt seine Aufgabe in Windeseile und erzeugt ein Compilat mit ebensolchen Eigenschaften.

Wenn ein C64-Besitzer diese Geschwindigkeiten bestaunt, sollte er aber selbstverständlich daran denken, daß die Taktfrequenz des Z80-Prozessors im C128 doppelt so hoch wie die des 6510 ist und der Computer somit auch um einiges schneller arbeiten kann. Leider ist sie aber (effektiv) wiederum nur wenig mehr als halb so hoch wie bei anderen Z80-Computern, was bedeutet, daß Turbo-Pascal auf anderen CP/M-Systemen noch schneller ist.

Trotzdem muß man sagen, daß Turbo-Pascal der schnellste derzeit für den C128 verfügbare Pascal-Compiler ist.

Um eine Vorstellung davon zu bekommen, welche Geschwindigkeiten Turbo-Pascal bei der Arbeit entwickelt, sei hier zum Abschluß ein kleiner Geschwindigkeitstest aufgeführt.

Die Konkurrenten sind Oxford-Pascal und Profi-Pascal, die beide für den C64 erhältlich sind und natürlich Turbo-Pascal.

Es ist selbstverständlich klar, daß dies ein Rennen unter ungleichen Voraussetzungen ist, da man Programme für verschiedene Prozessoren nur schlecht miteinander vergleichen kann. Schon gar nicht, wenn es sich um einen Kampf zwischen dem 6510 und dem durch 16-Bit-Befehle viel leistungstärkeren Z80-Prozessor handelt. Dieser Vergleichstest soll auch nur zeigen, was mit einem schnellen Computersystem und einem wirklich guten Programm wie Turbo-Pascal geleistet werden kann. Auch für den C64-Anwender ist das sicherlich mal ganz interessant; für den C128-Benutzer wohl um so mehr, da ihn das vielleicht dazu bringt, das CP/M-System seines Computers etwas näher zu betrachten.

Die Aufgabe war für alle drei Pascal-Compiler die gleiche. Es mußte eine Integer-Variable von 0 bis 2000 hochgezählt und jede davon auf dem Bildschirm angegeben werden. Ein anderer Versuch wurde ohne Bildschirmausgabe gemacht. Außerdem wurde die Übersetzungszeit für jeden Compiler gemessen. Das Testprogramm ist in Bild 2 abgedruckt. Die Ergebnisse (Bild 3) zeigen Turbo-Pascal in nahezu allen Bereichen als glänzenden Sieger. Nur mit der Ausgabe auf dem Bildschirm haperte es ein bißchen. Da konnte ihm das wirklich schnelle Profi-Pascal den ersten Platz streitig machen, was aber weniger an Turbo-Pascal als an der sehr umständlichen und langsamen Bildschirmansteuerung im CP/M des Commodore 128 liegt. Betrachtet man aber die übrigen Zeiten, so wird es klar, daß Turbo-Pascal eindeutig der schnellste Compiler ist. Die Arbeitsgeschwindigkeit von Turbo-Pascal zeigt, was ein Z80-Prozessor alles vermag. Ein kleiner Anstoß vielleicht für manchen Interessierten, den Z80 näher kennenlernen zu wollen. Ganz sicher aber ein gutes Argument für die Qualität von Turbo-Pascal.

(M.Thomas/ev)

Turbo-Pascal ist ein eingetragenes Warenzeichen von Borland International. Vertrieb in Deutschland: Heimsoeth Software, Fraunhoferstr. 13, 8000 München 5, Preis 198 Mark. Markt & Technik Verlag AG, Hans-Pinsel-Str. 2, 8013 Haar bei München.



	Compilation	Ausführung mit Ausgabe	Ausführung ohne Ausgabe
Oxford-Pascal	3.7 sec	34.7 sec	2.3 sec
Profi-Pascal	24.8 sec (Disk)	25.1 sec	1.0 sec
Turbo-Pascal	0.5 sec	30.6 sec	0.4 sec

Bild 3. Pascal-Compiler im Vergleich. Die Ergebnisse des Geschwindigkeitstests.

Plus

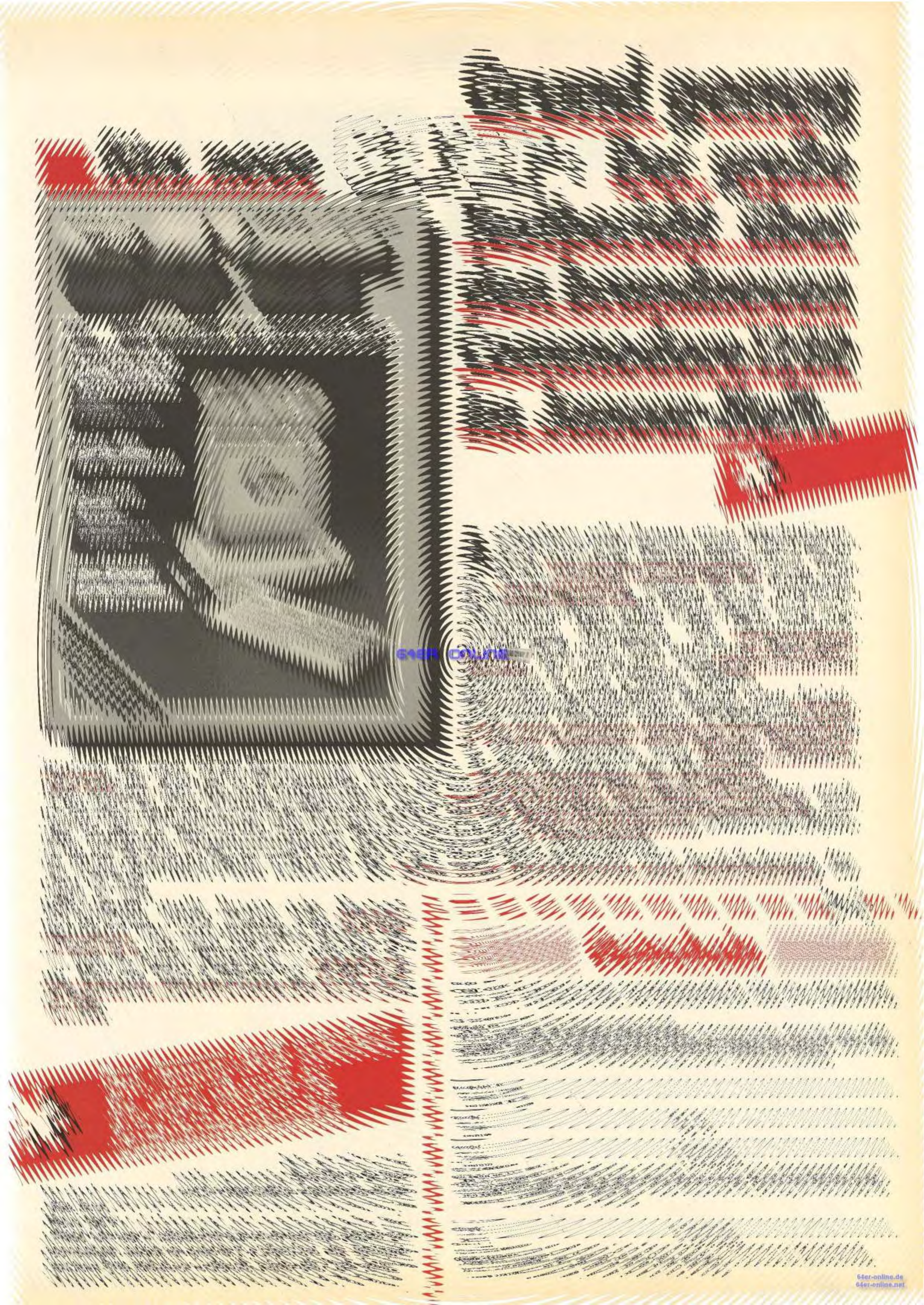
- Sicherheitskopie kann vom Besitzer angefertigt werden
- Editor mit den Qualitäten eines Textverarbeitungssystems
- Editor-Kommandos WordStar-kompatibel
- Extrem schneller Compilations-Vorgang
- Hohe Geschwindigkeit des übersetzten Programms
- Sehr komfortable Fehlerkorrektur
- Einbindung von Maschinensprache möglich
- Automatische Overlay-Technik
- Hoher Bedienungskomfort und hohe Bediensicherheit
- Wirth-Standard wird unterstützt
- Viele sinnvolle zusätzliche Standardprozeduren und Funktionen
- Standardtyp String mit entsprechenden Funktionen implementiert
- Direkter Zugriff auf CP/M-System-Ebene und CPU-Register
- Hohe Genauigkeit bei reellen Zahlen (11 Stellen)
- Hoher Programmierkomfort durch viele Compiler-Optionen
- Automatisches Finden von Laufzeit-Fehlern
- Systemmeldungen editierbar
- Ausführliches, gut verständliches Handbuch
- Sinnvolles, größeres Beipielprogramm im Lieferumfang
- Unterstützung der Turbo-Pascal-Käufer durch den Hersteller
- Gemessen an der Leistung sehr preiswert

Minus

- Kein Compilerprotokoll auf Drucker möglich

```
program geschwindigkeitstest;
   var i : integer;
   begin
     writeln('Dies ist ein
     Geschwindigkeitstest');
     for i:= 0 to 2000 do
           writeln(i,'   ');
           (* Diese Zeile wurde
           beim 2. Versuch
           wegelassen *)
     writeln;
     writeln('Fertig');
   end.
```

Bild 2. Das Pascal-Programm zum Geschwindigkeitstest











Multiplan

Das Trainingsbuch zu Multiplan beginnt mit der Erklärung der leeren Bildschirmmaske. Anschließend werden alle Befehle ausführlich besprochen und mit vielen kleinen Beispielen verdeutlicht. So lernt der Leser schrittweise, Felder anzulegen und mit ihnen zu arbeiten. Testfragen nach jedem Kapitel dienen der Selbstkontrolle und fassen die wichtigsten Informationen zusammen. Erst zum Schluß des Trainingsbuches werden mit Multiplan eine etwas umfangreichere Kostenübersicht und eine Fahrzeugkalkulation angelegt. Hauptsächlich erfolgen die Erklärungen für die englische Version, teilweise wird auf Besonderheiten der deutschen Version hingewiesen. Ein kleines Manko ist nur darin zu sehen, daß die auf vielen Seiten abgedruckten Muster zum Zusammenlegen erst fotokopiert werden sollen. Ein Faltblatt oder ein verkleinerter Druck wären übersichtlicher. Bei der Kurzübersicht über alle Befehle mit knapper Erklärung vermißt man Verweise auf die Kapitel, in denen der Befehl ausführlich dargestellt wurde. Warum muß man dafür erst noch im Inhaltsverzeichnis, wo allerdings die Seitenangaben fehlen, nachsehen?

Im Gegensatz zum Trainingsbuch werden in »Multiplan richtig eingesetzt« zehn praktische Beispiele, wie Fakturierung, Fertigungsplanung oder Umsatzanalyse benutzt, um den Leser mit Multiplan gründlich vertraut zu machen, so daß er später eine Anwendung schreiben kann. Jedes der zehn unabhängig voneinander zu bearbeitenden Beispiele beginnt mit der Beschreibung des Vorhabens, der die Festlegung der Arbeitsschritte und die Listen der verwendeten Funktionen und Befehle folgen. So hat man schnell eine Übersicht. Anschließend wird jeder einzelne Schritt sehr genau erklärt. Besonders wichtige Passagen sind durch »Beachten« kenntlich gemacht. Schiefgehen kann bei solcher Anleitung, die sich auf die englische Version von Multiplan beschränkt, eigentlich nichts. Für die praktische Arbeit sollte aber noch ein Stichwortverzeichnis aufgenommen werden.

Info: Dietmar Froitzheim »Das Trainingsbuch zu Multiplan« Data Becker 1984, 250 Seiten, ISBN 3-89011-016-9, 49 Mark

Robert E. Williams »Multiplan richtig eingesetzt« Markt & Technik Verlag AG 1983, 211 Seiten, ISBN 3-922120-30-X, 58 Mark

Commodore 128 intern

Mit diesem runde 500 Seiten umfassenden Werk zum C 128 stellt Data Becker dem C 128-Anwender wieder einmal erfreulich schnell ein Standardwerk zur Verfügung, das eine Unzahl an Informationen enthält, die woanders derzeit nicht zu bekommen sind. Das Buch bietet umfassende Information über

- den VIC-Chip (Grafik, Sprites, Zeichensatz, Farbe)
- die Ein- und Ausgabesteuerung (Allgemeines über die CIAs, Echtzeituhr, serieller Bus)
- den Sound-Chip SID (Programmierung, A-D-Wandlung, Filter, Synchronisation und Ringmodulation)
- den 8563 VDC-(80-Zeichen-) Chip (die Register, Zeichensatz, Cursor-Modus, hochauflösende Grafik)
- die Memory Management Unit MMU (Speicherkonfiguration, Betriebsarten, Zero Page in andere Speicherbereiche legen)
- die Assembler-Programmierung (Nutzung von Kernel-Routinen, Umgang mit verschiedenen Speicherbänken, Unterprogramme in anderen Speicherbänken aufrufen)
- das ROM des C 128 (Zero Page, Kernel-Routinen, Basic-Token, Tastaturmatrix, Zeichensätze, Z80ROM, kommentiertes ROM-Listing)
- die Hardware (CPU, Adreßlogik, RAMs und ROMs, 40- und 80-Zeichen-Hardware)

Die Kapitel über Ein-/Ausgabesteuerung, VIC- und Sound-Chip bieten gegenüber den entsprechenden Artikeln in entsprechenden C 64-Büchern nichts Neues, sind aber für den Neu-Einsteiger dennoch von großem Interesse und gehören auch einfach der Vollständigkeit halber in ein umfassendes Werk über den C 128.

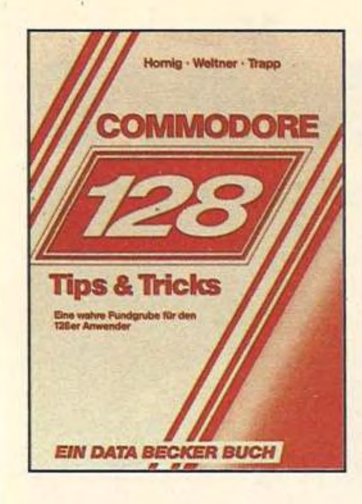
Commodore 128 Tips & Tricks

Das Buch ist, wie im Untertitel versprochen, wirklich »eine wahre Fundgrube für den 128er Anwender«. Mit einer Einschränkung: Viele der in diesem Buch veröffentlichten Programme, Tricks und Hinweise sind aus dem entsprechenden Data-Becker-Buch zum C 64 übernommen und allenfalls geringfügig überarbeitet worden. Wer also schon über genügend Literatur zum C 64 verfügt, dem bietet dieses Buch also nur in den C 128-spezifischen Teilen etwas Neues. Diese Teile des Buches (vom Umfang her der weitaus überwiegende Teil) haben es denn allerdings in sich. Einige ausgewählte Abschnittsüberschriften mögen dies verdeutlichen:

- Der 80-Zeichen-Bildschirm (Zuverlässiger Video-RAM-Zugriff, sinnvolle Manipulation des VDC 8563, hochauflösende Grafik, Manipulation des Bildschirmformats)
- Grafik mit den eingebauten Befehlen (Torten- und Balkengrafik, Funktionsplotter, Windows, Sprite-Handling)
- Nützliche Programme (OLD, Musik nebenbei, Analoguhr, modifiziertes INPUT)
- Software-Schutz auf dem C 128 (Zeilennummern-Roulette, Schutz durch POKEs, künstliche Steuerzeichen, Kopierschutz für Disk)
- Rund um die Tastatur (Ändern der Tastaturbelegung, Nutzung der Hilfstasten, Hextastatur für den C 128)
- Befehlserweiterung - selbst gemacht (wie verändert man die CHRGET-Routine, mehrere zusätzliche Befehle)
- Banking (Theoretische Grundlagen, Umschalten der Banks über die MMU)
- Autostart mit Floppy und Modulen
- Nützliche Speicheradressen
- Verwendbarer Speicher für Maschinenprogramme













- Wechseln des Betriebssystems
- Der 64er-Modus auf dem C 128

Man sieht, das Buch bietet ganz in der Tradition der Tips & Tricks-Bücher von Data Becker eine große Vielfalt an Informationen, die sowohl dem Einsteiger als auch dem Fortgeschrittenen das Arbeiten mit seinem Computer erleichtern. Viele Dinge, die in keinem Handbuch stehen, aber von großem Interesse für den Anwender sind, werden an Beispielen und Demo-Programmen erklärt. (ev)

Ralf Hornig, Tobias Weltner, Jens Trapp: Commodore 128 Tips & Tricks, Data Becker 1985, 328 Seiten, ISBN 3-89011-097-5, 49 Mark.

CP/M und WordStar Anwender-Handbuch

Dieses Buch bietet auf 120 Seiten eine leichtverständliche Einführung in das Betriebssystem CP/M. Quasi als Zugabe werden auch noch die Grundlagen von MP/M (CP/M für Multi-User-Anwendungen) und WordStar (Textverarbeitung) besprochen.

Leider ist dieses Buch wie praktisch alle anderen Bücher zum Thema CP/M mit einem Mangel behaftet: Es beschäftigt sich nur mit dem CP/M-Betriebssystem bis zur Version 2.2. Die beim C 128 eingesetzte CP/M-Version 3.0, auch CP/M Plus genannt, wird mit keinem Wort erwähnt. Das ist zwar angesichts des Erscheinungsdatums der ersten Auflage (1981) verständlich, aber dennoch ärgerlich für den CP/M 3.0-Anwender, gibt es doch einige Unterschiede zwischen den Versionen 2.2 und 3.0.

Das soll allerdings nicht heißen, daß der C 128-Besitzer mit diesem Buch nichts anfangen könnte. Denn die grundlegenden Eigenschaften des CP/M-Systems sind ja gleich geblieben. Zudem hat sich die Version 3.0 nie richtig durchsetzen können, so daß es kaum 3.0-Software gibt. Aufgrund der Aufwärtskompatibilität zwischen den CP/M-Versionen laufen auch CP/M 2.2-Programme ohne Einschränkungen auf dem C 128, so daß der stolze Besitzer des neuen Commodore so manche wertvolle Information über das CP/M-System aus diesem Buch ziehen kann.

Die Ausführungen über das Multi-User-System MP/M darf man allerdings getrost vergessen. Der C 128-Benutzer wird damit nie im Leben in Berührung kommen, es sei denn, er besucht irgendwann ein Museum für Betriebssysteme, die sich nicht durchsetzen konnten.

Der letzte Teil des Buches bringt eine Einführung in das Textverarbeitungsprogramm WordStar. Die wichtigsten Kommandos zur Handhabung werden etwas ausführlicher erklärt, weniger gebräuchliche in tabellarischer Form nebst Kurzerklärung zusammengestellt.

Alles in allem eines der Bücher, die auf der neuen CP/M-Welle schwimmen. Als Einstiegslektüre wohl durchaus brauchbar, für den ernsthaft an den CP/M-Fähigkeiten des Commodore 128 Interessierten aber denn doch wohl nur eine unnütze Geldausgabe. (ev)

Info: R. Paul, M. Riedel: CP/M und WordStar Anwender-Handbuch, Markt & Technik-Verlag AG, ISBN 3-922120-10-5, 120 Seiten, 29 Mark

Software-Schnellkurs dBase II

dBase II ist ein sehr bekanntes Datenbankprogramm. Weniger bekannt ist aber, wie man das Programm handhaben muß. Offenbar fehlen vielen potentiellen Anwendern Zeit und Geduld, sich mit dem (guten) Handbuch die notwendigen Fertigkeiten anzueignen.

Mit dem vorliegenden Schnellkurs lernt der Leser sehr schnell, eigene Dateien mit Records (Datensätze) und Feldern (die Angaben im Record) anzulegen, auszuwerten und Änderungen durchzuführen. Zusätzlich erfährt der Leser, wie er die Datei ausdrucken und seine mit dBase II erstellte Adreßdatei an Mailmerge übergeben kann. Ohne tiefgehende Begründungen werden jeweils komplette Beispiele mit allen notwendigen Schritten vorgestellt. So lernt der Leser in ganz kurzer Zeit das Prinzip des Datenbanksystems kennen. Es war allerdings nicht die Absicht des Autors, alle Möglichkeiten von dBase II aufzuzeigen, für eine schnelle Einführung ist das Buch aber gut geeignet. Den im zweiten Teil des Untertitels versprochenen Nutzen als Nachschlagewerk kann das Buch deshalb nur bescheidenen Ansprüchen erfüllen. (D. Hein/ev)

Info: Wolfgang Maaß, »Software-Schnellkurs dBase II«, Markt&Technik Verlag AG 1984, 104 Seiten, ISBN 3-922120-58-5, 37 Mark.

Software-Schnellkurs WordStar

Dieser 80 Seiten umfassende Band macht seinem Untertitel alle Ehre: »Was man für den Umgang mit WordStar wissen muß« heißt es da. Und das ist es genau, worum es hier geht. Dieses Buch eignet sich sowohl für den Einsteiger, der zum erstenmal mit WordStar in Kontakt kommt, als auch für den fortgeschrittenen Anwender, der nur ab und zu nochmal etwas nachschlagen möchte. Und eben für dieses »kurze Nachschlagen« ist das Buch gemacht. Denn wer wälzt schon gerne das mehrere hundert Seiten umfassende Original-WordStar-Handbuch, nur um sich Klarheit über die Funktion eines bestimmten Kommandos zu verschaffen?

Natürlich kann dieses Buch nicht alle möglichen Fragen zu WordStar erschöpfend klären. Doch das ist auch gar nicht Sinn und Zweck der Schnellkurs-Reihe. Dafür findet man alle ständig wieder benötigten Funktionen übersichtlich und mit Beispielen erläutert. Die spezielle Spiralheftung ermöglicht es, das Buch aufgeschlagen neben dem Computer liegen zu haben und ab und zu etwas nachzuschlagen – und das ist schließlich auch der eigentliche Sinn dieses Buches. (ev)

Info: Wolfgang Maaß: Software-Schnellkurs WordStar, Markt & Technik Verlag AG 1984, ISBN 3-922120-56-3, 80 Seiten, 37 Mark

Der etwas andere C64

Im Zusammenhang mit dem C64-Modus des C128 wird immer von »Vollkompatibilität« geredet. Daß dem doch nicht so ist, beweisen wir Ihnen hier.



»Ist der C128 wirklich vollkompatibel zum C64 ?« fragten uns schon viele Leser. Nach Lektüre dieses Artikels darf man zumindest nicht mehr mit ruhigem Gewissen »Ja« sagen sondern nur ein »Ja, aber...«. Denn einige Speicherstellen sind etwas anders belegt als beim C64. Wir zeigen Ihnen im folgenden, welche Speicherstellen das sind, welche Folgen das hat und welche Programme deswegen nicht laufen.

Der neue Videochip

Im C128 befindet sich für die 40-Zeichen-Darstellung eine modifizierte Version des VIC-Chips aus dem C64. Es sind nämlich zwei neue Register hinzugekommen. Die Speicherzelle 53295 dient zur Tastaturabfrage. Ein POKE 53295, 248 schaltet zum Beispiel die Zehnertastatur und die zusätzlichen Tasten in der oberen Reihe wieder ein. Allerdings erhält man nur »Müll« auf dem Bildschirm, wenn man eine der Tasten drückt. Grund dafür ist, daß der Editor im C64 mit den Tastaturcodes dieser Tasten überhaupt nichts anzufangen weiß. Ein Programm, mit dem sich diese Tasten auch im C64-Modus »vernünftig« benutzen lassen, finden Sie in diesem Sonderheft.

Viel interessanter ist aber die zweite Speicherstelle 53296. Dort sind die ersten beiden Bits belegt. Das Bit 0 steuert die Taktfrequenz des Systems. Damit läßt sich also der C64 auch unter 2 Megahertz fahren, also doppelt so schnell wie üblich. Allerdings kommt der Video-Chip dann mit dem Bildaufbau nicht mehr nach: Auf dem Bildschirm erscheint nur wirres Rauschen, als ob ein Fernseher auf einen Kanal ohne Programm eingestellt worden wäre. Das zweite Bit, Bit 1, schaltet den Bildschirm und die Farberzeugung vollkommen aus. Man sieht also nur einen leeren Bildschirm, der entweder schwarz, weiß oder grau ist, je nach eingestellter Randfarbe.

Zusammengefaßt:

POKE 53296,1 : 2 Megahertz einschalten

POKE 53296,2 : Bildschirm und Farbe aus

POKE 53296,3 : 2 Megahertz und Bildschirm aus

POKE 53296,0 : alles normal

»Schön und gut«, werden manche jetzt sagen, »Der neue Videochip hat halt ein paar zusätzliche Register. Deswegen läuft doch die Software, oder?«. Ja, 99,99 Prozent laufen. Aber Ausnahmen bestätigen die Regel: Das Spiel »Rescue on Fractalus« von Lucasfilms/Activision setzt zum Beispiel aus Versehen das Bit für den 2 Megahertz-Modus – und ein C128 im C64-Modus verabschiedet sich mit weißem Rauschen. Andere Fälle sind uns noch nicht bekannt, können aber durchaus noch vorhanden sein.

Der deutsche Zeichensatz

Das zweite Kompatibilitätsproblem ist der zweite eingebaute Zeichensatz. Im C64-Modus kann man ihn ja durch Drücken der »Caps Lock«-Taste ein- und wieder ausschalten. Das Einschalten geht allerdings auch softwaremäßig – und das wird manchen Programmen zum Verhängnis. Der deutsche Zeichensatz wurde, gelinde gesagt, etwas wüst im Computer »verdrahtet« und wird vom Prozessorport, der hauptsächlich für Kassettenoperationen und Memory-Mapping verantwortlich ist, mitgesteuert. Zur Verdeutlichung: Mit POKE 0,PEEK(0) OR 64 setzt man ein Bit im Steuerregister, das die Leitung, die den zweiten Zeichensatz steuert, von Ein- auf Ausgabe umschaltet. Mit POKE 1,55 wird nun der deutsche Zeichensatz eingeschaltet und läßt sich nicht mehr über die Taste »Caps Lock« abschalten! Hier hilft nur POKE 1,119, der wieder den Normalzustand herstellt.

Das wäre ja im Prinzip nicht allzu schlimm. Doch beim C64 schalten die meisten Programme mit POKE 1,55 das Basic-ROM ein, und nicht mit POKE 1,119, wie es auf dem C128 im C64-Modus nötig wäre. Sollte nun aus irgendwelchen Gründen das eine Bit in Speicherzelle 0 gesetzt sein, hat man den deutschen Zeichensatz auf dem Bildschirm und kriegt ihn so schnell nicht mehr da weg. Beim deutschen Zeichensatz fehlen einige Grafik-



zeichen, die durch die Umlaute ersetzt worden sind, so daß bei Programmen, die die Grafikzeichen verwenden, schon mal Seltsames auf dem Bildschirm entstehen kann. Dies geschieht zum Beispiel bei dem Platinenlayout-System »Platine 64« von Data Becker und bei einigen Spielen, die den Original-Commodore-Zeichensatz verwenden (»Jumpjet«).

Außerdem scheint dies irgendwelche Auswirkungen auf den Kassettenbetrieb zu haben. So können manche Originalprogramme aus England beim besten Willen und korrekt eingestellter Datasette nicht geladen werden. Die Turbo-Lader der Kassetten hängen sich an irgendeiner Hardwareänderung auf, wir können nur vermuten, daß es sich hier ebenfalls um den angezapften Prozessorport handelt. Probleme gab es namentlich mit einigen wenigen Programmen der Firmen »Ocean« und »U.S. Gold«. Allerdings handelte es sich hier auch nur um Ausnahmefälle.

Geisterbilder

Und noch ein Letztes stimmt nicht zwischen C 64 und C 128 im C 64-Modus überein. Der C 64 hat im $D-Bereich sogenannte Geisterbilder der einzelnen Bausteine. Als Beispiel: Der Video-Chip sitzt normalerweise auf $D000. Allerdings befindet sich an den Adressen $D100, $D200 und $D300 nochmals der Video-Chip, das heißt: POKEt man einen Wert nach $D100, befindet er sich in $D000, etc. Ähnlich geht es mit dem Sound- und den I/O-Chips. Beim C 128 sind aber nicht alle Geisterbilder vorhanden, so steht bei $D600 der 80-Zeichen-Video-Chip. Dort ist beim C 64 ein Geisterbild des Sound-Chips. Wir kennen zwar kein Programm, das auf diese Geisterbilder zugreift, es wären allerdings welche denkbar – und die würden auf dem C 128/C 64 natürlich abstürzen.

Also ist der C 128 im C 64-Modus nicht vollkompatibel zum C 64. Die Kompatibilität beschränkt sich auf 99,95 Prozent der lieferbaren Software. Das rechtfertigt dann aber vollkommen den Begriff der Kompatibilität, denn kompatibel heißt ja

Und was nun?

nicht identisch. Man kann sogar sehr froh sein, daß so wenige Änderungen an so unüblichen Stellen vorgenommen wurden. Denn es ist wie gesagt nur ein sehr geringer Bruchteil der Software betroffen, andererseits passen nun alle Software-Hersteller auf, daß ihre Programme nicht auf diese kleinen Tricks hereinfallen. Außerdem versprachen auf Anfrage die meisten Hersteller schon für die nächsten Wochen neue Versionen der betroffenen Programme, die dann einwandfrei auf dem C 128 im C 64-Modus laufen werden. Und wem die 99,95 Prozent jetzt nicht reichen, der kann ja immer noch auf den bewährten C 64 zurückgreifen, der volle Kompatibilität zu sich selbst verspricht.

(bs)

Das ist der C 128

Mehr als ein C 64 und mehr als ein CP/M-Computer – der C 128 ist angetreten, die Marktführerschaft von Commodore im Homecomputer-Markt zu sichern.

Nicht jeder Computer, der PC heißt, ist ein Personal Computer. Der C 128 PC zum Beispiel ist noch eine ganze Menge mehr. Auf ihm laufen nämlich neben Wordstar und dBase II auch noch Summer Games und Ghost Busters. Er vereinigt damit zwei für Commodore-Homecomputer ungewöhnliche Eigenschaften in sich: Er ist voll kompatibel zu einem anderen Commodore Computer, nämlich zum Commodore 64 , und er ist dank seines Z80A-Zweitprozessors uneingeschränkt CP/M-fähig.

Eigentlich besteht der C 128 daher aus drei Computern auf einer Platine, nämlich einem C 64, dann natürlich einem C 128 und schließlich noch einer CP/M-Maschine. Dementsprechend existieren drei grundsätzliche Betriebsarten, in denen der C 128 jeweils in einer anderen internen Hardware-Konfiguration läuft. Nach dem Einschalten ohne Steckmodul und CP/M-Diskette im angeschlossenen Laufwerk befindet sich der Computer im normalen C 128-Modus. Die Einschaltmeldung auf dem für den C 128 entwickelten 1901-Monitor versetzt jeden C 64-Besitzer in ehrfürchtiges Staunen: »122365 Bytes Free« heißt es da; gewiß nicht alltäglich für einen 8-Bit-Computer. Der Bildschirm hat im Textbetrieb eine Organisation von 40 Zeichen mal 25 Zeilen. Darüber hinaus kann im C 128- und im CP/M-Modus auch eine Darstellung mit 80 Zeichen mal 25 Zeilen (auf RGB- oder monochromem Monitor) gewählt werden.

Der C 128 kann die 16 Grundfarben des C 64 darstellen: in den Betriebsarten C 64 und C 128 dazu auch noch die bekannten 8 Sprites. Für den C 64-kompatiblen Betrieb ist das Basic 2.0 enthalten, für den CP/M-Betrieb wird das Betriebssystem CP/M 3.0 plus von der Diskette geladen.

Das Gehäuse des C 128 ist im modernen und eleganten ultraflachen Styling gehalten und hebt sich wohltuend vom klobigen Einerlei der VC 20/C 64/C 16-Gehäuse ab. Die Tastatur macht insgesamt einen soliden Eindruck (Bild 1).



Zusätzlich zur normalen Schreibmaschinentastatur gibt es zur schnellen Zahleneingabe einen numerischen Tastenblock mit 14 Tasten.

Der deutsche Zeichensatz wird durch die Umschalttaste ASCII/DIN erreichbar. Die Belegung der Tasten entspricht dann den DIN-Normen. Die Bezeichnungen sind bereits auf den Tastenkappen aufgedruckt. Durch diesen Zeichensatz ergibt sich teilweise eine fünffache Belegung einzelner Tasten. Doch diese Tatsache wird durch speziell auf den deutschen Markt zugeschnittene Textverarbeitungsprogramme und andere professionelle Software gerechtfertigt.

Auffällig sind vier farblich abgesetzte Viererblocks von Tasten oberhalb von Schreibmaschinentastatur und Ziffernblock. Ganz rechts handelt es sich dabei um die von allen Commodore-Heimcomputern bekannten Funktionstasten, die nach dem Einschalten mit Basic-Befehlen belegt sind (Tabelle 1). Im C 64-Modus können die Funktionstasten jedoch aus Gründen der Kompatibilität nur in der gewohnten Art und Weise vom Anwender-Programm abgefragt werden.

Links neben diesem Block finden sich die vier Cursortasten. Die restlichen acht Sondertasten sind mit zum Teil sehr speziellen, aber nützlichen Funktionen belegt.

Die ESC-Taste hat die Bedeutung wie beim C 16/Plus 4. Gefolgt von einer Buchstabentaste führt ESC eine Reihe von Sonderfunktionen wie Bildschirm-Scrollen, zeilenweises Löschen oder Cursor positionieren aus (Tabelle 2).

Die Alt-Taste hat im normalen Betrieb keine Bedeutung, kann aber bei entsprechender Programmierung von Anwenderprogrammen aus benutzt werden, um anderen Tasten eine neue Bedeutung zu geben (ähnlich wie bei der Control-Taste).

Die Line-Feed-Taste wirkt wie »Shift Return« beim C 64; der Cursor springt an den Anfang der nächsten Bildschirmzeile, ohne daß ein Befehl ausgeführt würde. Durch Drükken der Taste »No Scroll« wird ein

F1	GRAPHIC	Grafik-Modus ein
F2	DLOAD"	Programm laden
F3	DIRECTORY	Directory anzeigen
F4	SCNCLR	Bildschirm löschen
F5	DSAVE"	Programm speichern
F6	RUN	Programm starten
F7	LIST	Programm listen
F8	MONITOR	Maschinensprache-monitor aktivieren

Tabelle 1. Die Funktionstastenbelegung

Taste	Funktion
A	Insert-Modus ein
B	Untere rechte Ecke eines Windows definieren
C	Insert-Modus aus
D	Bildschirmzeile löschen
E	Cursor-Blinkmodus aus
F	Cursor-Blinkmodus ein
G	Akustisches Signal aus
H	Akustisches Signal ein
I	Neue Bildschirmzeile einfügen
J	Cursor an Zeilenanfang setzen
K	Cursor an Zeilenende setzen
L	Bildschirm-Scrolling ein
M	Bildschirm-Scrolling aus
N	Normal-Modus 80-Zeichen Bildschirm
O	Insert,-Anführungs,- und Invers-Modus aus
P	Bildschirmzeile bis Cursor-Position löschen
Q	Bildschirmzeile ab Cursor-Position löschen
R	Invers-Modus 80-Zeichen-Bildschirm
S	Block-Cursor ein
T	Obere linke Ecke eines Windows definieren
U	Strich-Cursor ein (nur bei 80 Zeichen)
V	Rollt Bildschirm um eine Zeile nach oben
W	Rollt Bildschirm um eine Zeile nach unten
X	Umschaltung 40/80 Zeichen und zurück
Y	Voreingestellte Tabulatorstops setzen
Z	Alle Tabulatorstops löschen
@	Bildschirm ab Cursor-Position löschen

Tabelle 2. Die ESC-Funktionen beim C 128

Listing angehalten; bei beliebigem Tastendruck geht's dann weiter.

Die Help-Taste ist eine sinnvolle Hilfe bei der Fehlersuche. Falls ein Programm mit Fehlermeldung abbricht, reicht ein Druck auf die Help-Taste, um die fehlerhafte Zeile am Bildschirm aufzulisten. Der Teil der Zeile, in dem der Fehler aufgetreten ist, wird dabei blinkend dargestellt.

Sehr wichtig ist die 40/80 Zeichen-Taste. Wie bei »Shift-Lock« handelt es sich um eine einrastende Taste. Je nachdem, in welcher Stellung sich diese Taste beim Einschalten oder bei einem Reset befindet, geht der C 128 entweder in den 40- oder in den 80 - Zeichen-Modus. Die 80-Zeichen-Darstellung ist natürlich nur im C 128-Modus oder unter CP/M möglich. Sind 80 Zeichen gewählt, dann wird der vom C 64 übernommene VIC II Video Chip, der auch beim C 128 für Bildschirmdarstellung, Grafik und Sprites sorgt, einfach abgeschaltet.

Doch keine Angst, der Bildschirm bleibt nicht dunkel, denn jetzt übernimmt ein vorsorglich eingebauter zweiter Video-Chip mit dem prosaischen Namen 8563 die Kontrolle. Dieser Baustein ist nämlich im Gegensatz zu dem in erster Linie aus Kompatibilität zum C 64 eingebauten VIC II in der Lage, 80 Zeichen pro Zeile zu kontrollieren.

An Anschlüssen verfügt der C 128 über einen User-Port, einen Datasetten-Port, einen Modulatorausgang, einen Audio-Eingang, einen Video-Ausgang, einen digitalen RGB-Ausgang, einen seriellen Port zum Anschluß der Commodore-Peripherie-Geräte sowie über zwei Joystick-Ports (Bilder 2 und 3).

Die ebenfalls neuen Diskettenlaufwerke 1570/1571 wie auch das beim Modell C 128/D integrierte Laufwerk ist in den Betriebsarten C 64 und C 128 kompatibel mit der Floppy 1541. Durch doppelseitige Benutzung der Diskette ergibt sich allerdings bei der 1571 eine Speicherkapazität von maximal 360 KByte (formatiert) pro Diskette. Das Double-Density, Double-Sided (doppelte Aufzeichnungsdichte - beidseitige Aufzeichnung)-CP/M-Format erlaubt eine Speicherkapazität von bis zu 410 KByte. Die Übertragungsrate zwischen Diskettenlaufwerk und C 128 beträgt im C 64-Modus wie gewohnt 300 Byte pro Sekunde. Im C 128-Modus beträgt die Übertragungsrate bereits immerhin 1500 Zeichen pro



Bild 2. Der PC 128 von hinten gesehen. Von rechts nach links: User-Port, RGB-Ausgang, Fernseher, Composite, Video, Serieller Port, Datasetten-Anschluß, Expansions-Port für Steckmodule



Bild 3. Anschlüsse und Schalter an der rechten Seite: Netzteilanschluß, Einschaltknopf, Reset-Taster, Joystick-Ports 2 und 1





Sekunde, liegt also etwa im »Hypra-Load«-Bereich.

Super-Basic 7.0

Beim Basic-Interpreter zeigt sich der C 128 und der C 128D ohne Zweifel von einer seiner stärksten Seiten: Das Basic 7.0 enthält alle Befehle und Funktionen der Basic-Versionen 2.0 (C 64), 3.5 (C 16 und Plus/4) und 4.0 (CBM 80xx). Damit stehen bereits leistungsfähige Grafikbefehle wie DRAW, BOX oder CIRCLE sowie viele Diskettenkommandos zur Verfügung. Doch damit nicht genug. Zusätzlich enthält das 7.0-Basic eine Reihe spezieller Befehle zur Steuerung von Sprites und zur einfachen Programmierung des Synthesizer-Bausteins (SID). Die zusätzlich zum 2.0-Basic vorhandenen Befehle und Funktionen sind in Tabelle 3 beschrieben.

Schon eine erste, oberflächliche Betrachtung dieser Tabelle läßt eine neue Dimension der Basic-Programmierung erahnen. Endlose DATA-Orgien und wüster GOTO-Dschungel gehören mit diesem Basic endgültig der Vergangenheit an. Formatierte Zahlenausgabe mittels PRINT USING ist dabei ebenso selbstverständlich wie Befehle zur Abfrage von Joystick, Lightpen und Paddels.

Mit WINDOW läßt sich ein Bildschirmfenster definieren, auf das sich anschließend alle PRINT- und INPUT-Befehle beziehen. Der Befehl mit dem beziehungsreichen Namen SLEEP läßt den C 128 denn auch tatsächlich für die angegebene Zeit schlafen: »SLEEP 5« hält das Programm fünf Sekunden lang an. So spart man sich das umständliche Hantieren mit leeren FOR...Next-Schleifen für oftmals sinnvolle Verzögerungen im Programmablauf. Zeiten zwischen einer Sekunde und 18 Stunden (!) sind programmierbar, womit sich die Frage aufwirft, wer seinen Computer während eines Programmes wohl für mehr als eine Minute anhalten will.

Eine Reihe von Befehlen dient ausschließlich der bequemeren Programmentwicklung: AUTO gibt bei der Programmeingabe automatisch die Zeilennummern vor, mit TRON kann in der Testphase eines Programmes eine Trace-Funktion eingeschaltet werden. Es werden dann auf dem Bildschirm die Zeilennummern der gerade abgearbeiteten Basic-Zeilen angezeigt. Dies bewährt sich insbesondere bei Fehlern, in der Programmlogik.

AUTO	Automatische Zeilennummerierung
APPEND	Öffnet eine sequentielle Datei zum Datenanfügen
BACKUP	Kopiert eine komplette Diskette
BANK	Wählt Speicherbank für PEEK, POKE und SYS
BEGIN...BEND	Faßt mehrere Basic-Zeilen zu einem Block zusammen
BOOT	Lädt und startet CP/M von Diskette
BOX	Zeichnet Rechtecke
BSAVE	Speichert beliebige Speicherbereiche auf Floppy
BUMP	Liefert bei Sprite-Kollisionen die Sprite-Nummer
CATALOG	Listet Inhaltsverzeichnis der Diskette
CHAR	Fügt Text in hochauflösende Grafik ein
CIRCLE	Zeichnet Kreise, Ellipsen und Vielecke
COLLECT	Löscht offene Dateien und reorganisiert Diskette
COLLISION	Dient zur Sprite-Kollisions-Abfrage
COLOR	Setzt Farben für Text und Grafik
CONCAT	Verbindet zwei sequentielle Dateien miteinander
COPY	Kopiert eine Disketten-Datei
DCLEAR	Schließt alle Kanäle zur Diskettenstation
DCLOSE	Schließt Kanal zur Diskettenstation
DEC	Dezimalwert einer Hexadezimalzahl
DELETE	Löscht einen Zeilenbereich aus dem Programm
DIRECTORY	Disketteninhaltsverzeichnis (wie CATALOG)
DLOAD	Lädt ein Programm von Diskette
DOPEN	Öffnet Kanal zur Diskettenstation
DO...LOOP	Programmschleife LOOP springt immer zu DO zurück.
DRAW	Setzt Punkte und zeichnet Linien
DSAVE	Speichert ein Programm auf Diskette
DS	Ergibt den Fehlerstatus des Diskettenlaufwerks
DS$	Enthält Fehlerstatus der Floppy im Klartext
DVERIFY	Überprüft Programmspeicherung auf Disk
EL	Enthält Zeilennummer bei Auftreten eines Fehlers
ELSE	Alternative bei IF-THEN, falls Bedingung nicht erfüllt
ENVELOPE	Definiert Hüllkurve für Synthesizer
ER	Liefert den Code des zuletzt aufgetretenen Fehlers
ERR$	Liefert Fehlermeldung im Klartext
EXIT	Dient zum Verlassen einer DO...LOOP-Schleife
FAST	Schaltet auf doppelte Geschwindigkeit (2MHz Takt)
FETCH	Holt Daten aus beliebiger Speicherbank (RAM-Floppy)
FILTER	Setzt den Klangfilter-Parameter für den SID
GETKEY	Wartet auf Tastendruck
GO64	Schaltet in den C 64-Modus
GRAPHIC	Wählt Grafik-Modus aus
GSHAPE	Schreibt ein Shape aus einem String auf den Bildschirm
HEADER	Dient zum Formatieren von Disketten
HELP	Listet nach Fehlermeldung die Fehlerzeile am Bildschirm auf
HEX$	Wandelt Dezimalzahlen in Hexadezimal-Strings
INSTR	Ergibt Position eines Teilstrings in einem anderen String
JOY	Fragt Joystickposition ab
KEY	Dient zur Belegung der Funktionstasten
LOCATE	Positioniert den Grafik-Cursor
MID$	Ermöglicht jetzt auch Wertzuweisung an Teilstrings
MONITOR	Ruft den eingebauten Maschinensprache-Monitor auf
MOVESPR	Bewegt ein Sprite über den Bildschirm
PAINT	Füllt einen Bereich der hochauflösenden Grafik aus
PEN	Fragt Lightpen ab
PLAY	Spielt die in einem String abgelegte Tonfolge
POINTER	Ergibt die Adresse einer Variablen im Speicher
POT	Fragt Paddles ab
PRINT USING	Erlaubt formatierte Zahlenausgabe
PUDEF	Definiert Steuerzeichen für PRINT USING.
RCLR	Liefert gewählten Farbcode für Text und Grafik
RECORD	Positioniert Schreib-/Lesezeiger bei relativen Dateien
RENAME	Dient zum Umbenennen von Diskettendateien
RENUMBER	Numeriert das Basic-Programm neu
RESTORE	Setzt DATA-Zeiger auf beliebige Zeilennummer
RESUME	Rückkehr aus einer Fehlerbehandlungsroutine
RGR	Liefert die Nummer des eingestellten Grafikmodus
RREG	Weist Variablen die Werte der Prozessorregister zu
RSPRCOLOR	Liefert den aktuellen Code des Mehrfarben-



	modus für Sprites
RSPPOS	Liefert Position und Geschwindigkeit eines Sprites
RSPRITE	Ergibt je nach Parameter alle Sprite-Attribute
RWINDOW	Liefert Parameter des eingestellten Bildschirmfensters
SCALE	Maßstabswahl bei hochauflösender Grafik
SCNCLR	Löscht Text- oder Grafikbildschirm
SCRATCH	Löscht eine Diskettendatei
SSHAPE	Speichert ein Shape in eine Stringvariable
SLEEP	Hält die Programmausführung für eine wählbare Zeit an
SLOW	Schaltet von 2 MHz auf 1 MHz Takt zurück
SOUND	Erzeugt Toneffekte mit wählbarer Frequenz und Dauer
SPRCOLOR	Setzt Mehrfarben-Modus-Farben für Sprites
SPRDEF	Ruft den integrierten Sprite-Editor auf
SPRITE	Setzt Sprite-Attribute
SPRSAV	Speichert ein Sprite in einem String oder umgekehrt
STASH	Überträgt Daten in eine Speicherbank (RAM-Floppy)
SWAP	Tauscht Daten zwischen zwei Speicherbänken aus
TEMPO	Setzt Abspieltempo für PLAY-Anweisung
TRAP	Verzweigt im Fehlerfall zu einer Fehlerbehandlungsroutine
TROFF	Schaltet Programmablaufverfolgung (Trace) aus
TRON	Schaltet Trace ein
UNTIL	Setzt Bedingung für DO...LOOP fest (DO UNTIL...)
VOL	Setzt Lautstärke für die SOUND-Anweisung
WHILE	Setzt Bedingung für DO...LOOP fest (DO WHILE..)
WIDTH	Setzt die Strichstärke für alle Grafikbefehle
WINDOW	Definiert ein Bildschirmfenster
XOR	Liefert die Exklusiv-Oder-Verknüpfung zweier Werte

Tabelle 3. Diese Befehle sind im Basic 7.0 dazugekommen. Die Befehle von Basic 2.0 (C64/VC 20) sind nicht aufgeführt, aber dennoch voll im Basic 7.0 integriert.

Eine falsch gesetzte IF-Abfrage wird damit zum Beispiel schnell erkannt – man sieht ja, wohin das Programm springt. Zu Testzwecken kann TRON natürlich auch im Programm verwendet werden. Am Anfang eines »verdächtigen« Programmteils fügt man einfach den TRON-Befehl ein, am Ende dieses Abschnittes wird die Trace-Funktion mit TROFF wieder außer Betrieb gesetzt.

Der RENUMBER-Befehl dient zum Umnumerieren des gesamten Programms oder auch nur einzelner Teile davon. Während jedoch RENUMBER beim bekannten Simons-Basic für den C 64 weder GOTO- noch GOSUB-Adressen ändert (und mithin eher ein Problem als ein Hilfsmittel darstellt), korrigiert das 7.0-Basic automatisch alle Zeilennummern hinter GOTO, GOSUB, THEN, ELSE, RESTORE und RESUME und sogar bei Abfragen von Fehlerzeilen mittels der Spezialvariablen EL in einer Fehlerbehandlungsroutine. Wobei wir gleich bei einem weiteren interessanten Aspekt des 7.0-Basic wären.

Fehlerbehandlung ohne Programmabbruch

Während der C 64 bei jedem auftretenden Fehler unerbittlich sein Programm mit einer entsprechenden Meldung beendet, bietet der C 128 hier einiges mehr an Flexibilität. Mit der TRAP-Anweisung können alle auftretenden Fehler während des Programmablaufs abgefangen werden. Zum Beispiel wird nach der Anweisung »TRAP 500« beim Auftreten eines Fehlers das Programm nicht unterbrochen, sondern es wird in eine Fehlerbehandlungsroutine (hier ab Zeile 500) verzweigt. Alle wichtigen Daten über den Fehler werden in Systemvariablen gespeichert und können von der (vom Programmierer zu schreibenden) Basic-Routine ab Zeile 500 ausgewertet werden: EL enthält die Zeilennummer, in der der Fehler auftrat, ER enthält die Fehlernummer und ERR$ liefert die Fehlermeldung im Klartext. Die Fehlerbehandlungsroutine kann diese Variablen auswerten, um gezielte Maßnahmen zu ergreifen. Anschließend sollte das Programm natürlich weiter fortgesetzt werden können. Dazu dient die RESUME-Anweisung, die eine Fehlerbehandlung abschließt (vergleichbar mit RETURN bei Unterprogrammen). RESUME kann auf drei verschiedene Arten verwendet werden. RESUME ohne weitere Parameter kehrt zu der Anweisung zurück, die den Fehler verursacht hat und setzt das Programm dort ganz normal fort. In diesem Falle muß natürlich in der Fehlerbehandlungsroutine die Fehlerursache behoben worden sein, sonst tritt der Fehler sofort wieder auf. Ein gutes Beispiel ist der Test, ob der Drucker eingeschaltet ist:

```
10 TRAP 90 : OPEN 1,4
20 PRINT#1,"DRUCKER OK"
30 END
90 IF ER=5 AND EL=10 THEN PRINT
"BITTE DRUCKER EINSCHALTEN UND
TASTE DRUECKEN" : GETKEY A$
95 RESUME
```

Dieses kleine Demo-Programm gibt den Text »Drucker OK« auf einem angeschlossenen Drucker aus. Falls der Drucker nicht eingeschaltet sein sollte, würde der OPEN-Befehl in Zeile 10 normalerweise zur Fehlermeldung »Device not present« führen. Diese Meldung wird aber durch den TRAP-Befehl im Falle eines Falles abgefangen und statt dessen zur Zeile 90 verzweigt, wo nach Überprüfung auf Fehlernummer und -zeile der Benutzer höflich aufgefordert wird, doch bitteschön den Drucker einzuschalten. Der Befehl GETKEY wartet anschließend auf einen Tastendruck, worauf das Programm durch den RESUME-Befehl wieder zum OPEN-Kommando zurückkehrt.

Soll das zum Fehler führende Kommando nicht nochmals ausgeführt werden, dann muß die Fehlerbehandlungsroutine mit RESUME NEXT abgeschlossen werden, wodurch mit dem nächsten Befehl nach der Fehlerursache weitergemacht wird. In besonderen Fällen kann es nach einem Fehler nützlich sein, ganz woanders im Programm fortzufahren. In einem solchen Falle kann hinter RESUME eine Zeilennummer angegeben werden, an der das Programm fortgesetzt werden soll.

Mit diesen Möglichkeiten zur Fehlerbehandlung im Programm selbst steht dem Programmierer ein leistungsfähiges Werkzeug zur Verfügung. Und sollte in der Entwicklungsphase eines Programmes doch einmal ein Fehler auftreten, dann genügt ein Druck auf die HELP-Taste, um die fehlerhafte Zeile aufzulisten. Der Teil der Zeile, der den Fehler

verursacht hat, wird dabei revers dargestellt.

Natürlich lassen sich auch von der Diskettenstation gemeldete Fehler in ähnlicher eleganter Weise abfragen. Statt umständlich die Zeile 1 OPEN 1,8,15: INPUT#1,A,B$,C,D: PRINT A,B$, C,D : CLOSE 1: END einzugeben (und dabei womöglich sein Programm zu überschreiben) tippt man beim 7.0-Basic einfach »?DS$« und erhält die gleiche Meldung. Die Systemvariable DS$ enthält nämlich den Fehlerstatus der Diskettenstation als Klartext, die Systemvariable DS den entsprechenden Fehlercode.

Überhaupt stehen beim C 128 alle Diskettenbefehle als Basic-Kommandos zur Verfügung. SCRATCH beispielsweise löscht ein File von der Diskette, DIRECTORY oder CATALOG listen das Inhaltsverzeichnis ohne Programmverlust, mit DLOAD, DSAVE und DVERIFY spart man sich das lästige »,8«. BLOAD und BSAVE dienen zum Laden/Speichern beliebiger Speicherinhalte (Maschinenprogramme, Grafik etc.). Neu sind auch eine Reihe von Befehlen zur komfortablen Verwaltung sequentieller und relativer Dateien. Mit RECORD kann beispielsweise direkt auf einen Datensatz einer relativen Datei zugegriffen werden, APPEND ermöglicht das Anfügen weiterer Datensätze bei sequentiellen Dateien.

Programmieren ohne GOTO

Zwei weitere ungewöhnliche Befehle fallen sofort auf, nämlich SLOW und FAST. Mit diesen Befehlen kann der C 128 zwischen 1 MHz Taktfrequenz (SLOW) und 2 MHz umgeschaltet werden. Nach dem Einschalten läuft der Computer mit einem Takt von 1 MHz, also mit ähnlicher Geschwindigkeit wie der C 64. Durch die komplizierte Art der Speicherverwaltung mit den verschiedenen Speicherbänken für Programme, Variablen und Betriebssystem/Basic ist das C 128-Basic prinzipiell geringfügig langsamer als das C 64-Basic. Dies wird jedoch, wie schon erwähnt, einerseits durch den wesentlich leistungsfähigeren Befehlssatz mehr als aufgewogen, zum anderen kann durch den FAST-Befehl die Abarbeitungsgeschwindigkeit exakt verdoppelt werden.

So schön das im Pinzip auch ist, die Geschwindigkeitsvorteile des FAST-Modus muß man sich mit dem bereits erwähnten Nachteil erkaufen.

Das 7.0 Basic bietet eine ganze Reihe spezieller Schleifen- und Strukturbefehle zur GOTO-freien, strukturierten Programmierung. Da wäre zunächst einmal die Erweiterung der IF...THEN-Abfrage um die ELSE-Klausel. Bisher mußte man beispielsweise alternative Entscheidungen wie folgt programmieren:

```
10 IF A$="N" THEN PRINT "NEIN" :
GOTO 30
20 PRINT "JA"
30 REM HIER GEHT'S WEITER.
```

Im 7.0 Basic reicht dazu eine Zeile, und die ist noch um einiges leichter verständlich:

```
10 IF A$="NEIN" THEN PRINT
"NEIN":ELSE PRINT "JA"
```

Wenn A$ gleich »N« ist, dann wird »nein« gedruckt, sonst »ja«.

Leider ist die ELSE-Anweisung in dieser Form auf eine Zeile beschränkt. Abhilfe schafft hier die Klammerung mit BEGIN... BEND.

Alle zwischen BEGIN und BEND stehenden Basic-Zeilen stellen einen Block dar, der vom Basic-Interpreter genauso wie eine einzelne Zeile behandelt wird. Deshalb wird BEGIN...BEND besonders vorteilhaft bei IF-Abfragen benutzt:

```
10 INPUT "HEISST DEIN COMPUTER
COMMODORE ODER SCHNEIDER?";C$
20 IF C$="COMMODORE" THEN BEGIN
30:     PRINT "C 128 KAUFEN!"
40: BEND: ELSE BEGIN
50:     PRINT"VERRÄTER!"
60 BEND
```

Man beachte, daß sich die IF-Anweisung insgesamt von Zeile 20 bis Zeile 60 erstreckt. In diesem Beispiel erhält man den Ratschlag, sich einen C 128 zu kaufen, falls der Computer »Commodore« heißt. Hat man jedoch »Schneider» (oder etwas anderes) als Namen angegeben, wird man sofort als »Verräter« tituliert.

Natürlich können derartige IF...THEN...ELSE-Abfragen mit BEGIN...BEND auch geschachtelt werden, das heißt, man kann sowohl in den THEN- als auch in den ELSE-Teil weitere IF-Abfragen einbauen.

Somit lassen sich auch größere Programmblöcke ohne GOTO programmieren. Der Verzicht auf GOTO erhöht nicht nur die Übersichtlichkeit, sondern auch die Geschwindigkeit beim Programmlauf. Bei jedem GOTO-Befehl muß der Basic-Interpreter nämlich erstens die Zeilennummer, die im Programm ja als Dezimalzahl steht, in das interne binäre Format umrechnen und zweitens dann auch noch die angegebene Zeile suchen. Ein weiterer Vorteil: In den Programmbefehlen selbst kommen keine weiteren Zeilennummern mehr vor, das Beispielprogramm kann unverändert in allen möglichen Zeilenbereichen laufen.

Aber nicht nur Verzweigungen lassen sich derart elegant programmieren, besonders bei Schleifen, also bei Wiederholungen von bestimmten Programmteilen, spielt das 7.0 Basic seine Stärken erst richtig aus. Es ist ja vom C 64 her bekannt, daß eine FOR...NEXT-Schleife um einiges schneller ist als die gleiche Schleife mittels IF und GOTO programmiert. Nachteilig bei der FOR...Next Schleife ist, daß die Anzahl der Schleifendurchläufe schon bei Eintritt in die Schleife bekannt sein muß. Dieser Nachteil wird durch die neue, schnelle DO...LOOP-Schleifenstruktur behoben. Wie FOR...NEXT umklammert auch DO...LOOP einen beliebig großen Programmteil. Die Wirkung des DO-Befehls besteht einfach darin, daß der Basic-Interpreter sich den Anfang der Schleife »merkt«. Bei Erreichen des zugehörigen LOOP wird dann sehr schnell, ohne Suchzeiten, zu DO zurückgesprungen. Es ergibt sich also eine »unendliche Schleife« zwischen DO und LOOP. Um diese Schleife dennoch verlassen zu können, ist der EXIT-Befehl vorgesehen. Die Wirkung von EXIT besteht einfach darin, die Programmausführung hinter LOOP ganz normal fortzusetzen. Normalerweise wird EXIT daher von einer Bedingung abhängig gemacht. Beispiel:

```
10 X=1
20 DO
30 : X=X*2 : PRINT X
40 : IF X > 1500 THEN EXIT
50 LOOP
```

Der Wert X wird hier solange verdoppelt und ausgedruckt, bis der Wert 1500 überschritten wird. Neben dieser unbedingten DO...LOOP-Schleife sind noch zwei von Bedingungen abhängige Formen vorgesehen. DO WHILE... LOOP wird so lange ausgeführt, wie eine nach WHILE stehende Bedingung wahr ist:



```
10 DO WHILE A$="":GET A$:LOOP
```

Solange keine Taste gedrückt wird, ist A$ immer leer, die WHILE-Bedingung also erfüllt. Die Schleife wird daher erst verlassen, wenn eine Taste gedrückt wird.

Die DO UNTIL-Schleife wird dagegen nicht ausgeführt, solange die Bedingung wahr ist, sondern im Gegenteil so lange, bis die hinter UNTIL angegebene Bedingung wahr wird. Natürlich können auch bei DO WHILE oder DO UNTIL zusätzliche EXITs in die Schleife eingebaut werden, was die Leistungsfähigkeit dieser Anweisung noch erhöht.

Die Grafik ist für alle da

Um hochauflösende Grafik auf dem C 64 zu realisieren, gibt es außer dem Kauf diverser Basic-Erweiterungen (oder dem Abtippen von 64'er-Listings) im wesentlichen nur die Alternative, selbst zum Maschinensprache-Profi zu werden – ungefähr so, als wenn man Radio- und Fernsehmechaniker werden müßte, um an seinem Farbfernseher die Farbe einstellen zu können. Ein sicherlich unhaltbarer Zustand, dessen Änderung Commodore allerdings bereits mit dem 3.5-Basic des C 16 in Angriff genommen hatte. Die hochauflösende Grafik des C 128 ist genauso wie die des C 64/C 16 aufgebaut. Insgesamt 64000 Einzelpunkte können getrennt angesprochen werden, was einer Auflösung von 320 x 200 Punkten entspricht. Daneben ist ein Mehrfarbenmodus mit einer Auflösung von 160 x 200 Punkten vorgesehen, bei dem jeder Einzelpunkt eine von vier Farben haben kann.

Der große Unterschied zum C 64 liegt darin, daß die C 128-Grafik voll vom Basic unterstützt wird. Befehle wie DRAW, BOX oder CIRCLE ermöglichen schnelles und unkompliziertes Zeichnen geometrischer Figuren von Linien über Drei-, Vier- und Mehrecken bis hin zu Kreisen und Ellipsen. Alle Figuren können beliebig vergrößert, verkleinert und sogar gedreht oder ausschnittsweise dargestellt werden. PAINT füllt geschlossene Flächen aus, SCALE dient zur Skalierung der Zeichenfläche und SCNCLR löscht den Grafikbildschirm.

Alle Grafikbefehle arbeiten sowohl im Hochauflösungs- wie auch im Mehrfarben-Modus. Mit dem Befehl GRAPHIC wird der gewünschte Grafik-Modus eingestellt. Zur Wahl stehen Text mit 40 Zeichen pro Zeile, Hochauflösung, Hochauflösung mit Textfenster, Mehrfarbengrafik mit Textfenster und schließlich Text mit 80 Zeichen pro Zeile. Leider beziehen sich alle Grafikbefehle des 7.0 Basic ausschließlich auf die vom C 64 her bekannte 320 x 200 Punkte-Auflösung (und natürlich wahlweise auf den Mehrfarbenmodus mit 160 x 200 Punkten).

Doppelte Auflösung (640 x 200 Punkte) ist zwar im 80-Zeichen-Modus möglich, wird aber weder vom Basic noch vom Betriebssystem unterstützt. Dies ist um so ärgerlicher, als sich diese Fähigkeit mit wenig Aufwand hätte realisieren lassen. Das 64'er-Magazin brachte eine kleine Basic-Erweiterung zu diesem Thema in der Ausgabe 12/85 und zeigte Commodore damit, wie's geht. Wegen der großen Nachfrage finden Sie diese interessante Grafik-Erweiterung auch in diesem Sonderheft.

Ein weiterer Wermutstropfen: Die ganze schöne Grafik, Sprites und 40-Zeichen-Text sind ausschließlich über einen Composite-Monitor verfügbar, auf einem RGB-Monitor tut sich überhaupt nichts. Andersherum ist die 80-Zeichen-Textdarstellung nur über RGB (oder natürlich einen monochromen Monitor) möglich.

Der verblüffte Anwender stellt spätestens jetzt fest, daß er einfach einen Monitor zu wenig hat. Damit dürfte Commodore sich die Urheberrechte am ersten Zwei-Monitor-Heimcomputer der Welt gesichert haben. Wohlgemerkt, man hat nicht die Wahl zwischen Composite und RGB, sondern braucht unbedingt einen Composite-Monitor für 40-Zeichen, Grafik und Sprites und ebenso unbedingt entweder einen RGB- oder einen SW-Monitor für 80 Zeichen.

Abhilfe schafft hier der neue 1901-Monitor von Commodore (Bild 7), der speziell zum C 128 entwickelt wurde und sowohl über einen Composite- als auch über einen RGB-Eingang verfügt. Zwischen beiden Betriebsarten des Monitors wird mit einem kleinen Schalter an der Frontseite hin- und hergeschaltet – eine softwaremäßige Umschaltung ist nicht vorgesehen. Man kann daher nur wünschen, daß der Umschalter stabil genug gebaut ist – er wird oft betätigt werden müssen.

Als Fazit zur C 128-Grafik bleibt festzuhalten, daß sie von der Auflösung her dem durch den C 64 gesetzten Standard (320 x 200 Punkte) entspricht. Neu ist allerdings die vorbildliche Unterstützung durch das 7.0 Basic.

Wenn von Grafik die Rede ist, dürfen natürlich Shapes und Sprites nicht fehlen. Hinsichtlich dieser beweglichen Grafikobjekte ist beim C 128 eine gelungene Synthese von C 64-Hardware und C 16 Software zu verzeichnen. Vom C 64 stammen die acht Sprites, frei programmierbare, bewegliche Grafikobjekte, die von der Hardware (VIC-Chip) erzeugt und in den Bildschirm eingeblendet werden. Sprites können sowohl im 40-Zeichen-Textmodus als auch in den verschiedenen Grafik-Modi erzeugt werden, nicht allerdings im 80-Zeichen-Modus, denn der VIC, der sie erzeugt, ist nicht RGB-fähig.

Shapes und Sprites

Aus dem 3.5 Basic des C 16 wurde das Konzept der softwaremäßig erzeugten Shapes übernommen. Shapes sind rechteckige Ausschnitte aus der hochauflösenden oder der Mehrfarben-Grafik, die in Stringvariablen abgespeichert werden und daraus auch wieder auf den Bildschirm gebracht werden können. Da es sich um reine Grafikelemente handelt, können sie weder im 40- noch im 80-Zeichen-, sondern nur im Grafik-Modus dargestellt werden. Mit »SSHAPE X$,100,100,150,120« wird beispielsweise der Inhalt des Rechtecks mit linker oberer Ecke (100,100) und rechter unterer Ecke (150,120) aus der hochauflösenden Grafik in der Stringvariablen A$ abgelegt. Mit »GSHAPE A$,X,Y« wird die in A$ enthaltene Grafik-Information an der Grafikposition X,Y wieder auf den Bildschirm gebracht. Neben den Sprites sind die Shapes also eine zweite leistungsfähige Möglichkeit zur Darstellung grafischer Objekte und eröffnen in Zusammenhang mit der hohen Speicherkapazität des C 128 völlig neue Möglichkeiten für Spiele in hochauflösender Grafik.

Integrierter Sprite-Editor

Das Basic 7.0 enthält sogar einen integrierten Sprite-Editor, mit dem man direkt am Bildschirm das Punktmuster des gewünschten Sprites entwerfen kann. Mit dem SPRITE-Befehl werden für jedes Sprite folgende Attribute gesetzt: Aktivität, Farbe, Priorität, Dehnung in X-und Y-Richtung und Modus (hochauflösend oder Mehrfarben).

Mit «SPRITE 4,1,6,1,1,0,0« wird zum

Beispiel das Sprite Nr.4 aktiviert (1). Es wird in der Farbe Grün (6) angezeigt, hat Priorität über bereits angezeigte Bildschirmdaten (1), ist in X-Richtung gedehnt (1), in Y-Richtung nicht gedehnt (0) und wird im Hochauflösungs-Modus angezeigt (1).

Um umgekehrt die Attribute eines bereits definierten Sprites zu bestimmen, kann die RSPRITE-Funktion verwendet werden.

Mit dem SPRSAV-Kommando können die Daten eines Sprites in einer Stringvariablen abgelegt werden oder umgekehrt aus einem String ausgelesen werden.

Die Steuerung der Sprites erfolgt über den MOVESPR-Befehl mit dem ein Sprite an eine bestimmte Bildschirmposition gesetzt werden kann. Die Positionsangabe kann entweder in absoluten Koordinaten oder auch relativ zur bisherigen Position erfolgen. Doch damit noch nicht genug. Gibt man zusätzlich noch eine Geschwindigkeit als Zahlenwert zwischen 1 und 15 an, so gleitet das Sprite automatisch an die angegebene neue Position. Durch Setzen von Plus- oder Minuszeichen vor die Koordinatenangaben werden aus den absoluten Koordinaten relative Koordinaten. Ohne Geschwindigkeitsangabe erscheint das Sprite sofort an der neuen Position. »MOVESPR 7,-30,+40« versetzt Sprite 7 augenblicklich um 30 Punkte nach links und um 40 Punkte nach oben. Beim C 64 kann man durch PEEKen in die Sprite-Kollisionsregister des VIC feststellen, ob ein Sprite mit einem anderen Sprite oder mit Hintergrunddaten kollidiert ist. Beim C 128 bedient man sich für den gleichen Zweck um einiges eleganter der COLLISION-Anweisung. Damit kann eine automatische Programmunterbrechung bei Eintritt entweder einer Sprite/Sprite oder einer Sprite/Hintergrund-Kollision programmiert werden. »COLLISION 1,500« hat beispielsweise folgende Bedeutung: Falls im weiteren Verlauf des Programms eine Sprite-Sprite-Kollision (Kennziffer 1) auftritt, dann wird das laufende Basic-Programm unterbrochen, und es wird ein Unterprogramm ab Zeile 500 ausgeführt. Nach dem RETURN wird das Programm an der Unterbrechungsstelle fortgesetzt.

COLLISION und MOVESPR sind leistungsstarke Befehle, die ein Basic-Programm hinsichtlich der Sprite-Steuerung sehr stark entlasten. Um ein Sprite quer über den Bildschirm zu bewegen, muß man beim C 64 noch mit einer FOR...NEXT-Schleife arbeiten; um Kollisionen festzustellen, war daneben noch ein ständiges PEEKen in die Kollisionsregister des VIC nötig.

Beim C 128 reichen zwei Basic-Befehle, die zudem noch interruptgesteuert arbeiten, so daß das Basic-Programm während der Bewegung der Sprites weiterlaufen kann, bei einer eventuell auftretenden Sprite-Kollision dagegen automatisch unterbrochen wird, um schnell darauf reagieren zu können.

COLLISION dient nicht nur zur Sprite-Sprite und Sprite-Hintergrund-Kollisionsabfrage, sondern nebenbei auch noch zur Kontrolle eines Lichtgriffels, neuhochdeutsch auch als Lightpen bezeichnet. Um festzustellen, welches Sprite eine Kollision ausgelöst hat, verwendet man die BUMP-Funktion.

Zusammen mit vielen weiteren Befehlen zur Sprite- und Shape-Kontrolle sowie für hochauflösende Grafik ergeben sich natürlich ungeahnte Möglichkeiten für den Basic-Programmierer. Aber das ist nicht nur bei der Grafikprogrammierung der Fall.

Musikalisches Basic

Ein ähnlicher Komfort ist auch bei der Programmierung des aus dem C 64 übernommenen Synthesizer-Bausteins, des SID, zu finden. Alle Musik-Parameter müssen nicht mehr aus DATA-Wüsten in den SID hineingePOKEt werden, sondern können elegant und leichtverständlich per Basic-Befehl gesetzt werden.

VOL regelt zum Beispiel die Lautstärke, mit dem SOUND-Kommando wird einer der Tongeneratoren gestartet. Dabei kann über entsprechende Parameter nicht nur die Frequenz, sondern auch die Dauer des Tones sowie das An- und Abschwellen festgelegt werden.

Mit ENVELOPE wird jeweils eine von zehn möglichen Tonhüllkurven für Musikinstrumente definiert. Attack, Decay, Sustain, Release werden damit ebenso festgelegt wie Wellenform und Impulsbreite. Jede der zehn möglichen Hüllkurven bleibt gespeichert, bis sie durch einen weiteren ENVELOPE-Befehl zur gleichen Hüllkurvennummer überschrieben wird.

Nach dem Einschalten des C 128 sind bereits alle zehn Hüllkurven mit der Klangstruktur verschiedener Musikinstrumente vordefiniert: Klavier, Akkordeon, Zirkusorgel, Trommel, Flöte, Gitarre, Cembalo, Orgel, Trompete und Xylophon. Damit steht auch dem musikalisch wenig bewanderten Einsteiger sofort eine Fülle einfach anwendbarer Klangeffekte zur Verfügung. Mit der FILTER-Anweisung können zudem alle Filtermöglichkeiten des SID zur Klangverfremdung ausgeschöpft werden.

Musik per Warteschlange

Der Play-Befehl ermöglicht das automatische Abspielen von in Strings gespeicherten Musiknoten. In dem als Parameter angegebenen String können Informationen über Hüllkurve, Oktave, Lautstärke, Tonkanal und Filter enthalten sein, in der Hauptsache aber natürlich die zu spielenden Noten. Die Noten werden einfach durch Angabe des Notennamens (A,B,C,D,E,F,G) ausgewählt, wobei das B der in Deutschland üblichen Notenbezeichnung H entspricht. Natürlich können die einzelnen Noten um Halbtöne erhöht oder erniedrigt werden, es sind ganze, halbe, viertel, achtel und sechzehntel Noten, jeweils auch punktiert, möglich.

Auch der PLAY-Befehl wird interruptgesteuert ausgeführt, das heißt die zu spielenden Noten gelangen in eine Ton-Warteschlange, was nichts anderes bedeutet, daß sie in einem reservierten Speicherbereich abgelegt werden. Während des Interrupts stellt das Betriebssystem fest, ob Tondaten in der Warteschlange stehen. Wenn ja, wird der erste Wert aus der Schlange geholt und, vereinfacht gesprochen, an den Synthesizer-Chip (SID) zum Abspielen übergeben. Alle anderen in der Warteschlange stehenden Tondaten rücken jetzt einen Platz vor. Bei jedem weiteren Interrupt wird überprüft, ob die vorgesehene Tondauer bereits erreicht ist. Wenn dies schließlich der Fall ist, wird wieder nach wartenden Tondaten Ausschau gehalten, und das ganze Spiel setzt sich fort.

Wie gesagt, läuft dies alles jeweils während des System-Interrupts ab. Das Basic selbst »merkt« davon nichts. Es stellt nur fest, ob noch Platz in der Tonwarteschlange ist oder nicht. Falls noch Plätze frei sind, können weitere Tondaten angefügt werden und das Programm fährt anschließend normal fort, während die Musik automatisch abgespielt wird. Nur dann,



wenn zuviele Töne zum Abspielen anstehen, muß das Programm tatsächlich anhalten und warten, bis wieder Plätze in der Warteschlange frei geworden sind.

Obwohl Dank des leistungsstarken Basics nur selten nötig, gibt es natürlich auch beim C 128 den Zugriff auf die Maschinenebene. Allerdings ist es hier nicht einfach mit POKE, PEEK und SYS getan. Vielmehr ergibt sich aus dem Konzept der verschiedenen Speicherbereiche, die mittels Bank-Switching umgeschaltet werden, das Problem, in welche Speicherbank der POKE-, PEEK- oder SYS-Befehl gehen soll. Das C 128-Basic löst dieses Problem ebenso einfach wie elegant: Mit dem BANK-Befehl kann die gewünschte Speicherbank ausgewählt werden. Damit erfolgt natürlich nicht wirklich eine Bankumschaltung (während ein Basic-Programm läuft, muß natürlich immer der Basic-Interpreter eingeschaltet sein), sondern das Basic merkt sich nur, in welcher Speicherbank beispielsweise ein POKE-Wert abgelegt werden muß, oder aus welcher Bank die Daten für PEEK stammen müssen.

Die Verbindung zur Maschinensprache

So kann man nach Belieben entweder in den Programm- oder in den Variablenspeicher POKEn und PEEKen. Betriebssystem- und Basic 7.0-Routinen können nach »BANK 15« einfach mit SYS aufgerufen werden.

Daneben gibt es noch die Möglichkeit, von Basic aus ganze Speicherbereiche zwischen Bank 1 (Basic-Arbeitsspeicher) und anderen Speicherbänken hin- und herzuladen. Hierzu dienen die Befehle FETCH; STASH und SWAP. »FETCH 2000, 50000,4,35000« holt beispielsweise 2000 Byte ab Adresse 35000 aus Speicherbank 4 und legt diese ab Adresse 50000 im Basic-Arbeitsspeicher (immer Bank 1) ab. STASH ist die Umkehrfunktion zu FETCH: Es wird eine Anzahl Bytes aus dem Arbeitsspeicher in eine andere Speicherbank gebracht. SWAP schließlich tauscht die angegebenen Speicherbereiche in beiden Bänken gegeneinander aus.

Diese drei Befehle sind hauptsächlich für den Einsatz im Zusammenhang mit Speichererweiterungen (RAM-Floppy) gedacht.

Es können damit Datenmengen verwaltet werden, die ein mehrfaches von 64 KByte im Speicher belegen, und das mit Geschwindigkeiten, wie sie mit einer Floppy niemals zu realisieren sind.

Maschinensprache-Monitor eingebaut

Wem trotz allem die Möglichkeiten des 7.0 Basic noch nicht reichen, der kann mit dem MONITOR-Kommando das Basic verlassen und landet im fest im ROM eingebauten Maschinensprachemonitor. (Tabelle 4 zeigt den kompletten Monitor-Befehlssatz).

Befehl	Bedeutung
A(assemble)	Assembliert eine Zeile im 6502-Befehlscode.
C(compare)	Vergleicht zwei Speicherbereiche byteweise und zeigt Unterschiede an
D(disassemble)	Disassembliert einen Speicherbereich mit Maschinensprache-Code.
F(fill)	Füllt einen Speicherbereich mit einem angegebenen Byte.
G(go)	Startet ein Maschinensprache-Programm bei einer angegebenen Adresse.
H(hunt)	Durchsucht einen Speicherbereich nach einer angegebenen Bytefolge.
L(load)	Lädt eine Programmdatei von Kassette oder Diskette.
M(memory)	Zeigt den Inhalt eines angegebenen Speicherbereiches hexadezimal an.
R(register)	Zeigt den Inhalt der Prozessorregister an.
S(save)	Speichert den angegebenen Speicherbereich auf Kassette oder Diskette.
T(transfer)	Verschiebt den Inhalt eines angegebenen Speicherbereiches.
V(verify)	Vergleicht einen Speicherbereich mit einer Programmdatei auf Kassette oder Diskette.
X(exit)	Beendet den Monitor und kehrt in den Basic-Direktmodus zurück.
>	Modifiziert ein bis acht Speicherzellen ab der angegebenen Adresse.
.	Identisch mit dem A-Befehl
.	Modifiziert die Prozessorregister.
@	Zeigt den Floppy-Disk-Status an oder überträgt einen Floppy-Disk-Befehl.

Tabelle 4. Der Befehlssatz des eingebauten Maschinensprache-Monitors.

Dieser dem C 16-»Tedmon« nachempfundene Monitor enthält neben den üblichen Funktionen zum Listen und Beschreiben des Speichers und einem Disassembler auch einen kleinen Assembler, mit dem Maschinenprogramme sehr komfortabel eingegeben werden können. Statt der sonst üblichen vierstelligen hexadezimalen Adresseneingabe verlangt dieser Monitor allerdings fünf Stellen:

Die erste Stelle gibt an, welche von 16 möglichen Speicherbänken ausgewählt werden soll. Allerdings sind in der Grundversion des C 128 natürlich nicht alle 16 Bänke belegt, einige sind für ROM-Module, andere für die RAM-Floppy reserviert.

Der gesamte RAM-Bereich des C 128 umfaßt 128 KByte, aufgeteilt in zwei Banks zu je 64 KByte. RAM-Bank 1 enthält Zeropage, Stackbereich und Bildschirmspeicher; 60 KByte stehen nur für das Basic-Programm zur Verfügung. Der Speicherbereich für die Variablen befindet sich in RAM-Bank 2 und umfaßt 62 KByte. Doch damit noch nicht genug. In Stufen zu je 128 KByte kann der RAM-Bereich auf bis zu 512 KByte erweitert werden, wobei der zusätzliche Speicher als RAM-Disk angesprochen wird.

Der erreichbare RAM-Speicher in Zusammenhang mit dem wahlweise ladbaren CP/M 3.0-Betriebssystem beträgt ebenfalls 128 KByte. Auch hier läßt sich der Speicher mit der externen RAM-Disk-Option auf insgesamt 512 KByte erweitern.

Das Betriebssystem des C 128 belegt 16 KByte ROM für sich allein, das sehr umfangreiche Basic 7.0 ist in 32 KByte ROM untergebracht. Zu diesen 48 KByte ROM kommen noch weitere 16 KByte, die das Basic und das Betriebssystem des C 64 enthalten und die nur in der C 64-Betriebsart aktiv sind.

C 64-Steckmodule passen auch in den Modul-Port des C 128. Spezielle C 128-Steckmodule mit einer Kapazität bis zu 32 KByte können natürlich ebenfalls eingesteckt werden. Stellt das Betriebssystem nach dem Einschalten fest, daß ein C 64 Modul eingesteckt ist, dann wird automatisch zur entspechenden Betriebsart übergegangen und das Modul-Programm gestartet. Normalerweise gelangt man vom C 128

aber über den »GO 64«-Befehl in den C64-Modus.

Zur Verwaltung der diversen Speicherbänke bedient sich der C128 des sogenannten »Bank-Switching«. Bank Switching heißt soviel wie Speicherblock-Umschaltung. Die 128 KByte Speicher des C128 werden dazu in zwei Teile mit je 64 KByte gespalten. Mit einem Trick wird dafür gesorgt, daß der Prozessor abwechselnd die eine oder die andere 64-KByte-Bank »sieht«. Der Trick heißt Memory Management Unit (MMU). Wie der Name schon sagt, managt diese Schaltung die Speicherkonfiguration. Die MMU bestimmt, auf welche RAM-Bank der Prozessor »sehen« darf, also wo Schreib-/Lesezugriffe im Speicher erfolgen sollen. Aber nicht nur das. Die MMU gibt auch die ROM-Konfiguration an, sie sagt also dem Prozessor, aus welchem ROM er seine Befehle zu holen hat. Entweder aus dem Kernel oder aus einem EPROM einer Erweiterungskarte.

Im C128 wird der Basic-Speicher so verwaltet, daß die RAM-Bank 0 (64 KByte) für Basic-Programme und Bank 1 (64 KByte) für Basic-Variable reserviert ist. Für den Basic-Programmierer bedeutet das, daß er je etwa 60 KByte Speicher für das Programm und die Variablen zur Verfügung hat. Es ist nicht möglich, größere Programme auf Kosten des Variablenspeichers anzulegen. Die vollen 64 KByte pro Bank können auch nicht vollständig genutzt werden, da ein Bereich in beiden Bänken für die Zeropage, den Stack und den Bildschirmspeicher reserviert ist.

122365 Basic Bytes Free

Der Bereich geht bis $0400. Während ein Basic-Programm läuft, regelt die Memory Management Unit, auf welche Bank zugegriffen werden soll. Die Informationen darüber, ob gerade eine Variable oder Befehle zu verarbeiten sind, erhält die MMU vom Basic-Interpreter.

Aber nochmal zurück zu dem Bereich von $0000 bis $0400 der für das System reserviert ist. Die Besonderheit daran ist, daß in diesem Bereich nur Bank 0 existiert. Bank 1 kann dort nicht angesprochen werden. Auf diese Weise ist sichergestellt, daß der Prozessor immer auf dieselbe Bank zugreift und nicht deshalb abstürzt, weil in Bank 1 vielleicht ein anderer Stack steht als in Bank 0. Zusätzlich kann man sich selbst Bereiche von 1 bis 16 KByte in beiden Bänken reservieren, die am Speicheranfang oder Speicherende liegen können.

Man kann diesen Bereich beispielsweise in eigenen Maschinenroutinen als Stack oder Speicher verwenden, wenn zwischen den Bänken umgeschaltet werden muß und die gleichen Daten zur Verfügung stehen sollen.

Speicherzugriff erlaubt

Im Gegensatz zum C64 erlaubt der Expansion-Port des C128 einen direkten Speicherzugriff (DMA, Direct Memory Access). Direkter Speicherzugriff bedeutet, daß ohne Umwege über den Prozessor in den Speicher des C128 geschrieben oder der Speicher ausgelesen werden kann. Das wichtigste, um einen DMA realisieren zu können, ist, daß der Prozessor während des Zugriffs abgeschaltet bleibt. Beim C128 macht das der 8564-VIC. Er steuert den Daten- und Adreßbus so, daß der Prozessor und DMA sich nicht ins Gehege kommen, was beim C64 nicht immer sichergestellt ist. Bei einem gleichzeitigen Bus-Zugriff von Prozessor und externen Geräten erweist sich der Prozessor meist als der Schwächere, was zu ernsthaften Problemen führen kann.

Daß ein direkter Speicherzugriff ohne weiteres machbar ist, eröffnet dem C128 gegenüber dem C64 zusätzliche Einsatzgebiete in der Meßwerterfassung. Ein Meßgerät kann dadurch beispielsweise Meßwerte so schnell direkt in den Speicher schreiben, daß eine Echtzeiterfassung eines Meßvorganges möglich ist. Eine andere Möglichkeit wäre der Anschluß eines Festplatten-Laufwerks. Die Daten könnten dann viel schneller in den RAM-Bereich geladen oder aus dem Arbeitsspeicher geholt werden, als wenn der Prozessor vorher jedes Bit ein paar mal »umdreht«.

Der serielle Bus und der Kassetten-Port des C128 sind von den Anschlüssen her identisch mit denen des C64. Die Bedienung des seriellen Bus wurde überarbeitet, so daß der C128 zusammen mit den neuen Commodore Laufwerken 1570/71 wesentlich schneller speichern und laden kann als der C64.

Kompatibilität war schon immer ein Reizwort für Commodore. Deswegen war Skepsis angesagt, ob der C128 wirklich kompatibel zum C64 ist.

Also haben wir eine Zahl von Programmen ausprobiert, die direkt oder indirekt über einen Kopierschutz im Betriebssystem herumpfuschen oder sonstige Gemeinheiten anstellen, die jeden Nicht-C64 sofort zum Aussteigen bewegen würden. Erster Testkandidat war Hypra-Load. Einige Probeläufe zeigten, daß sich hier in Verbindung mit der 1541 überhaupt keine Probleme ergeben. Damit dürfte gesichert sein, daß alle Programme mit geänderten Busroutinen einwandfrei funktionieren.

Nächstes Testobjekt war ein Kopierprogramm, das intensiven Gebrauch von illegalen Opcodes macht. Mit Opcodes bezeichnet man den Befehlssatz des Prozessors. Illegale Opcodes sind Befehle, die der Hersteller des Prozessors eigentlich gar nicht vorgesehen hat. In Wirklichkeit bewirken aber manche von ihnen auch beim C64 schon etwas. Und einige Programme nutzen sie. Es hätte also sein können, daß der 8502-Prozessor einige dieser beim 6502 an sich undefinierten Opcodes benutzt. Doch traten hier keine Probleme auf. Auch alle anderen kopiergeschützten (und nicht kopiergeschützten) Diskettenprogramme konnten wir ohne Schwierigkeiten laden und benutzen.

GO 64 – wie kompatibel ist der C128?

Getestet wurden von uns diverse Spiele wie Ghostbusters und Pit Stop II. Auch hier ein eindeutiges Ergebnis: Grafik und Musik stimmen mit dem C64 überein. Von über 100 speziell für diesen Test ausgewählten Disketten-Programmen gab es nur bei einem Schwierigkeiten. Dies ist »Rescue on Fractalus, bei dem die Grafik vom C128 nicht richtig aufgebaut wird.

Auch die Datasetten-Besitzer dürften im allgemeinen keine Probleme mit ihren Programmen haben. Bei allen getesteten Kassetten-Programmen gab es nur in einem einzigen Fall Schwierigkeiten. Es handelt sich hierbei um das Spiel »Rolands Rat Race«.

Das Ansteuern von Drucker-Interfaces verlief ebenfalls problemlos. Akkustikkoppler beziehungsweise Modems, die über den User-Port angeschlossen werden, konnten einwandfrei betrieben werden. Keine Schwierigkeiten gab es auch bei dem Betrieb der Btx-Module von Astech und Commodore.

Alle für den C 64 vorhandenen Peripherie-Geräte, die über ein Interface angeschlossen werden, dürfen also auch weiterhin nutzbar bleiben.

Bei Modulen mit Modul-Start, die in den Expansion-Port eingeschoben werden, wird beim Einschalten des C 128 direkt in den C 64-Modus gesprungen. Andere Erweiterungen oder Geräte, die über den Expansion-Port angeschlossen werden (zum Beispiel der Ascom-Akkustikkoppler), beeinflussen den C 128-Modus nicht. Nach dem Einschalten gelangt man also in den C 128-Modus.

Nach unseren Tests können wir dem C 128 tatsächlich die Kompatibilität zum C 64 bestätigen. Doch der C 128 kann im C 64-Modus noch etwas mehr.

Im C 128-Modus kann man zwischen ASCII- und DIN-Tastatur über eine Umschalttaste oder über POKE-Befehle vom Programm aus wählen. Diese Umschaltung steht dem Benutzer auch im C 64-Modus zur Verfügung. Die DIN-Taste funktioniert auch im C 64-Modus. Allerdings wird im C 64-Modus nur der Zeichensatz, nicht aber die Tastaturcodes geändert. Dadurch kommt eine etwas merkwürdige Tastenbelegung zustande. Im Programm wird diese Umschaltung (ASCII nach DIN) folgendermaßen vorgenommen:

```
POKE 0, PEEK(0) OR 64: POKE1,0
```

Im C 128-Modus kann der Benutzer mit dem FAST-Befehl vom 1-MHz Takt auf den 2-MHz-Takt umschalten. Hierdurch kann eine erhebliche Steigerung der Verarbeitungsgeschwindigkeit erreicht werden. Dies ist auch im C 64-Modus möglich. Falls Sie bereits einen C 128 besitzen, geben Sie einmal folgendes Programm ein:

```
10 POKE 53265, PEEK (53265)
and 239
20 POKE 53296, PEEK (53296) OR 1
30 FOR I=1 TO 1000:PRINT"A":NEXT
40 POKE 53296, PEEK (53296) AND
254
50 POKE 53265, PEEK (53265) OR 16
```

Im Vergleich zwischen dem C 64 und dem C 128 im C 64-Modus mit 2 MHz wird die Ausgabe von 1000 mal »A« von 35 Sekunden auf 20 Sekunden verkürzt. Hierbei wird allerdings auch der Bildschirm abgeschaltet. Diese Routine kann für alle zeitaufwendigen Programmschritte genutzt werden, bei denen keine Bildschirmausgabe erfolgt.

Der dritte Unterschied zum C 64 taucht bei Benutzung der C 157 1-Floppy-Station auf. Im C 128-Mode wird die Diskette doppelseitig genutzt. Durch den Befehl

```
OPEN 15,8,15,"U0 >  M1":CLOSE 15
```

ist dies auch im C 64-Modus möglich. Formatieren Sie jetzt eine Diskette und lassen sich das Directory auflisten, werden Sie mit der Meldung »1328 Blocks free« überrascht. Sie haben also den Speicherplatz auf der Diskette verdoppelt.

Sind die Disketten schon einseitig beschrieben (zum Beispiel vom C 64) und man möchte die Rückseite nutzen, muß folgender Befehl eingegeben werden:

```
OPEN 15,8,15,"U0 >  H1"
```

Das Laufwerk gibt dann zwar ein starkes Rattern von sich, nutzt die Diskette aber von der Rückseite. Ist die Diskette neu formatiert, so unterbleibt das Rattern.

Der CP/M-Profi

Der C 128 enthält zwei Zentraleinheiten, nämlich einmal einen 8502-Prozessor (kompatibel zum 6510 und 6502), der sowohl im Normal- als auch im C 64-Modus aktiv ist, und einen Z80A-Prozessor (4 MHz), der unter CP/M die Kontrolle über den Computer übernimmt.

Schon beim Einschalten des C 128 macht sich der Z80 bemerkbar. Er versucht, das CP/M-Betriebssystem von Diskette zu booten (zu laden und zu starten). Ohne irgendeinen Befehl beginnt dabei das angeschlossene Diskettenlaufwerk zu laufen und versucht das Programm zu laden. Wird kein entsprechendes Programm gefunden, aktiviert der Z80 den 8502-Prozessor und der C 128-Modus wird eingeschaltet.

Geteilte Datenschienen

Beide Prozessoren, Z80 und 8502, sind in der Lage, miteinander zu kommunizieren, was auch im Betriebssystem vorgesehen ist. Reicht nämlich für bestimmt I/O-Operationen der BIOS-(Betriebssystem-)Befehlssatz des CP/M nicht aus, übernimmt der 8502 diese Aufgaben.

Z80 und 8502 teilen sich im C 128 die Adreß- und Datenleitungen. Da der Z80 schneller arbeitet als die übrigen Bausteine, paßt eine Interface-Schaltung die Geschwindigkeit des Z80 an das System an, was natürlich die Arbeitsgeschwindigkeit des Z80 verringert. Diese Interface-Schaltung sorgt dafür, daß der Z80 bei Buszugriffen nur mit 2 MHz anstelle der angegebenen 4 MHz getaktet wird. Das bedeutet, daß CP/M-Programme auf dem C 128 nicht ganz so schnell laufen, wie auf einem 4-MHz-CP/M-Computer.

Bleiben wir noch einen Moment beim CP/M. Es handelt sich dabei um eine neuere Version dieses 8-Bit-Betriebssystems für Z80-Prozessoren. CP/M 3.0 plus ist in der Lage, auch über 64 KByte hinausgehende Speicherbereiche einigermaßen sinnvoll zu verwalten.

Der C 128 wird einfach dadurch in den CP/M-Modus versetzt, daß man die CP/M-Boot-Diskette ins Laufwerk einlegt und den Computer einschaltet (oder einen Reset auslöst). Das Betriebssystem erkennt, daß es sich beim ersten Sektor der Diskette um einen Autostart-Sektor handelt (die ersten drei Byte eines Autostart-Sektors sind »CBM«), lädt und startet das in diesem Sektor vorhandene Maschinenprogramm automatisch. Dieses Programm stellt die erforderliche Speicherkonfiguration ein und lädt seinerseits Teile des CP/M-Systems nach. Dann geht die Kontrolle an den Z80A-Prozessor über, der das System schließlich vollständig lädt.

Das Diskettenformat unter CP/M ist kompatibel zu den gängigsten anderen Formaten, es können also beispielsweise CP/M-Disketten, die mit einem Kaypro oder Osborne CP/M-Computer geschrieben wurden ohne Probleme gelesen werden. Das Textverarbeitungsprogramm Wordstar als Testprogramm war beispielsweise auch in der Kaypro-Version nach kurzer Installation problemlos lauffähig. Damit ist für den C 128-Anwender die Tür zu professioneller Software weit aufgestoßen.

Wer bisher nur mit dem C 64 gearbeitet hat, dem stehen bei der Umstellung auf eine CP/M-fähige Maschine natürlich einige Umstellungen bevor.

Was ist CP/M?

Die Bezeichnung CP/M steht für »Control Program for Microcomputers«, also einfach für ein – inzwischen weit verbreitetes – Betriebssystem für Microcomputer. CP/M wurde bereits 1974 von Gary Kildall

bei Intel entwickelt und diente ursprünglich als Dateiverwaltungs-System für einen von Intel vertriebenen Compiler für die Programmiersprache PL/M. Seit 1975 wird CP/M als allgemeines Betriebssystem kommerziell angeboten und ständig weiterentwickelt. Das Erfolgsrezept von CP/M heißt »Standard«.

Da jeder Hersteller eines Mikrocomputers in der Regel sein eigenes Betriebssystem einbaut, ergeben sich sehr große Probleme bei der Übertragung von Programmen und Daten zwischen verschiedenen Computern. Wer schon einmal versucht hat, ein VC 20-Programm an den C 64 anzupassen (oder umgekehrt) der weiß, wovon hier die Rede ist.

Das CP/M-Betriebssystem tritt nun als Vermittler zwischen Anwenderprogramm und Hardware auf. Beispielsweise fragt ein CP/M-Programm niemals die Tastatur direkt ab, denn die kann ja infolge unterschiedlicher Hardware von Computer zu Computer anders konzipiert und angesteuert sein. Statt dessen ruft das Anwenderprogramm eine Routine im CP/M-System auf, die die gewünschte Funktion ausführt. Natürlich muß das CP/M-System für jeden Computer an die spezielle Hardware angepaßt werden, aber, und das ist das Entscheidende, die Anwenderprogramme laufen unverändert, da sie nicht direkt auf die Hardware zugreifen.

Da alle CP/M-Programme generell im gleichen Speicherbereich liegen, treten kaum Probleme bei der Übertragung von einer CP/M-Maschine zur anderen auf.

Grundvoraussetzung für die Übertragbarkeit von Software ist allerdings die Verwendung des gleichen Mikroprozessors in allen CP/M-Computern. Die Wahl fiel bei Intel nicht schwer: Der 8080 aus dem eigenen Hause wurde zum CP/M-Prozessor gekürt. Der Fairneß halber muß man allerdings zugeben, daß der 8080 damals (Mitte der siebziger Jahre) tatsächlich einer der leistungsfähigsten Mikroprozessoren überhaupt war.

Heutzutage wird allerdings fast ausschließlich der Z80 verwendet, der voll aufwärtskompatibel zum 8080 ist, aber über einen stark erweiterten Befehlsvorrat und mehr interne Register verfügt. Wer sich ernsthaft für CP/M interessiert, der sollte sich daher mit diesem Prozessor vertraut machen.

Im Gegensatz zur 6502-Prozessor-Familie (zu der auch der 6510 im C 64 und der 8502 im C 128 gehört) handelt es sich beim Z80 ebenso wie beim Vorgänger 8080 um eine sogenannte »registerorientierte CPU«. Das bedeutet, daß der Z80 über eine größere Anzahl interner Register verfügt, die sowohl als Daten- wie auch als Adreßregister benutzt werden können.

Der Z80-Prozessor

Insgesamt stehen dem Benutzer sechs allgemeine 16-Bit-Register (BC, DE, HL, BC', DE', HL') zur Verfügung, die wahlweise auch als zwölf 8-Bit-Register angesprochen werden können. Daneben gibt es zwei Akkus und ebenfalls zwei Flag-Register, zwei 16-Bit-Indexregister (IX und IY), einen 16-Bit-Stackpointer (SP) und natürlich einen Programmzähler (PC). Zwei weitere Register, das Interrupt-Vektor-Register I und das Refresh-Kontroll-Register R haben spezielle Funktionen.

Die Register AF, BC, DE und Hl bilden die sogenannten Primärregister, die vom 8080 übernommen wurden und die in erster Linie direkt angesprochen werden können. Mit speziellen Befehlen kann zwischen diesen Primärregistern und den »Sekundärregistern« AF', BC', DE' und HL' umgeschaltet werden. Die Indexregister IX und IY haben ähnliche Aufgaben wie die 6502-Register X und Y, sind jedoch 16 Bit breit.

Der Z80-Befehlssatz ist sehr umfangreich und umfaßt unter anderem auch 16-Bit-Arithmetik, Blockverschiebebefehle, automatische Such- und Ein-/Ausgabebefehle, bedingte Unterprogrammaufrufe sowie Einzelbitbefehle. Insgesamt kennt die Z80-CPU über 700 Opcodes. Um diese vielen Maschinenbefehle verschlüsseln zu können (mit einem Befehlsbyte können nur 256 Befehle codiert werden), gibt es einige spezielle »Umschaltopcodes«, die einfach bewirken, daß der Z80 intern auf eine andere Befehlstabelle umschaltet und das nächste Befehlsbyte in einer anderen Form interpretiert.

Nach diesem kurzen Ausflug zum Z80 jetzt jedoch wieder zum CP/M-Betriebssystem. Beim Studium von Handbüchern und CP/M-Literatur stößt man immer wieder auf vier Abkürzungen: BIOS, BDOS, CCP und TPA. Hinter diesen Bezeichnungen verbergen sich die wichtigsten Bestandteile eines CP/M-Systems.

BIOS steht für »Basic Input Output System«, hat aber absolut nichts mit der Programmiersprache Basic zu tun. Die Bezeichnung deutet lediglich an, daß es sich hierbei um eine Sammlung grundlegender Routinen zur Ein-/Ausgabe von Daten handelt.

Das CP/M-BIOS entspricht von der Funktion her ziemlich genau den Kernel-Routinen bei Commodore-Computern. Der gesamte Kontakt eines Anwenderprogramms mit der Hardware-Umgebung läuft über diese Routinen. Alle BIOS-Routinen werden indirekt über das BDOS, das Basic Disk Operating System, durchgeführt. Der Aufruf aller Systemroutinen geschieht durch einen Unterprogrammsprung zur Adresse 5, dem sogenannten BDOS-CALL. Die verschiedenen geforderten Funktionen werden dadurch selektiert, daß im C-Register des Z80 ein Funk-

Code	Funktion
0	System-Kaltstart
1	Zeichen von Tastatur holen
2	Zeichen auf Bildschirm ausgeben
3	Zeichen von externem Gerät holen
4	Zeichen an externes Gerät senden
5	Zeichen an Drucker senden
6	Direkte Ein-/Ausgabe über Konsole
7	I/O-Zuordnung holen
8	I/O-Zuordnung festlegen
9	String auf Bildschirm ausgeben
10	String von Tastatur einlesen
11	Feststellen, ob Taste gedrückt
12	Liefert Versionsnummer des CP/M-Systems
13	Reset des Diskettensystems
14	Laufwerk auswählen
15	Datei öffnen
16	Datei schließen
17	Datei auf Diskette suchen
18	Zweite Datei suchen
19	Datei löschen
20	Nächsten Block lesen
21	Nächsten Block speichern
22	Neue Datei erzeugen
23	Dateinamen ändern
24	Angesprochene Laufwerke feststellen
25	Aktuelles Laufwerk feststellen
26	DMA-Pufferadresse festlegen
27	Bit-Map-Adresse holen
28	Schreibschutz setzen
29	Schreibschutzinformation holen
30	Datei-Attribute setzen
31	File-Parameter-Adresse holen
32	USER-Nummer setzen
33	Daten von Diskette lesen
34	Daten auf Diskette schreiben
35	Dateigröße feststellen
36	Datensatzadresse holen

Tabelle 5. Die CP/M-Funktionsaufrufe

tionscode übergeben wird. Die Routine ab Adresse 5 verzweigt dann abhängig vom Inhalt des C-Registers zu den verschiedenen BIOS-Funktionen (Tabelle 5).

Diese beiden Teile des CP/M-Systems, BIOS und BDOS, werden beim »Booten«, also beim Laden des Systems von der Diskette, einmal in den Speicher geholt und bleiben dann ständig resident vorhanden.

Für den Kontakt zwischen CP/M und Anwender sorgt der »Console Command Processor« (CCP). Es handelt sich dabei um einen einfachen Kommando-Interpreter, der über die Tastatur eingegebene CP/M-Kommandos erkennt und ausführt. Mit CP/M-Kommandos hat es dabei eine besondere, grundlegende Bewandtnis. Hierzu ist es wichtig zu wissen, daß Dateinamen unter CP/M aus zwei Teilen bestehen, der eigentlichen Dateibezeichnung (acht Zeilen Länge) und der Namenserweiterung (»Extension«, drei Zeichen Länge). Diese Erweiterung, die durch einen Punkt vom eigentlichen Namen abgetrennt wird, gibt den Typ der Datei an.

Kommandostruktur

Besonders wichtig ist der Dateityp »COM«. Wenn ein Dateiname die Erweiterung COM hat, dann stellt sie ein CP/M-Kommando dar. Das funktioniert folgendermaßen: Wenn der Benutzer ein Kommando eingibt, dann sucht der CCP auf der Diskette nach einer COM-Datei, die den Namen des Kommandos trägt. Diese Datei wird dann geladen und gestartet, mit anderen Worten, die Datei ist das Kommando.

Will man beispielsweise das Directory der Diskette sehen, dann gibt man den Befehl DIR ein. Der CCP sucht, lädt und startet daraufhin das Programm »DIR.COM«, das wiederum das Inhaltsverzeichnis der Diskette ausgibt. Derartige Kommandos heißen »transiente Kommandos«, weil sie eben nicht fest eingebaut sind, sondern nur bei Bedarf geladen werden. Der für transiente Kommandos reservierte Speicherbereich heißt »Transient Program Area« (TPA), womit auch die vierte der oben angesprochenen Abkürzungen geklärt wäre. Jedes CP/M-System verfügt auf der Systemdiskette über eine Standard-Bibliothek von wichtigen transienten Kommandos zum Ändern von CP/M-Parametern, zum Kopieren, Löschen, Listen und Umbenennen von Dateien und ähnliches, was man zum sinnvollen Arbeiten mit einer Diskettenstation benötigt.

Natürlich ist es möglich, diesen Befehlsvorrat zu erweitern, indem man entsprechende Assemblerroutinen schreibt oder auch einen Compiler einsetzt. Einzige Bedingung für transiente Kommandos ist die Lauffähigkeit des Maschinenprogramms in der TPA. Generell sind alle CP/M-Anwenderprogramme transiente Kommandos. Das Textverarbeitungsprogramm Wordstar heißt unter CP/M zum Beispiel »WS.COM« und wird einfach mit dem Kommando »WS« aufgerufen.

Mit dem CP/M-Betriebssystem hat der C 128-Benutzer Zugriff auf das größte Softwareangebot der Welt. Immerhin werden bereits seit zehn Jahren CP/M-Programme entwickelt und immer weiter verbessert. Textverarbeitung, Datenbanken, Tabellenkalkulation stehen in großer Auswahl ebenso zur Verfügung wie spezielle Branchenlösungen für den kommerziellen Einsatz. Statistik-Pakete, mathematische und naturwissenschaftliche Software ist in Mengen erhältlich. Wer selber programmieren will, der findet unter CP/M ein Angebot an Programmiersprachen, das von Fortran und Cobol über Pascal bis zu Ada und Lisp reicht. Unter CP/M gibt es so ziemlich alles, was auch auf Großrechnern läuft.

Allerdings soll nicht verschwiegen werden, daß trotz Standard und Kompatibilität gewisse Probleme auftreten können. So muß CP/M-Software generell erst einmal an einem speziellen Computer angepaßt werden. Das erledigt ein zu jeder Software gehöriges sogenanntes Installations-Programm, das zum Beispiel die Tastaturbelegung, Bildschirmansteuerung etc. abfragt, um beispielsweise ein Textprogramm optimal auf den jeweiligen Computer einstellen zu können. Dieser kleine Arbeitsaufwand ist aber natürlich gar nichts im Vergleich beispielsweise zur Aufgabe, etwa den Textomat oder Vizawrite an den VC20 anzupassen.

Als Fazit darf festgehalten werden, daß dem C 128-Besitzer durch das CP/M-Betriebssystem ein riesiges Software-Potential zur Verfügung steht. Inzwischen gibt es auch bereits die erste »schlüsselfertige« CP/M-Software für den C 128: Wordstar, Turbo Pascal, dBase II und Multiplan gibt es fix und fertig an den C 128 angepaßt. Alle vier Programme sind zu Preisen unter 200 Mark erhältlich – geradezu sensationell, wenn man sich die Preise sonstiger CP/M-Software ansieht. Es dürfte von entscheidender Bedeutung sein, ob auch andere Hersteller von CP/M-Software willens sind, die Preise für ihre Produkte, die sich teilweise noch im vierstelligen Bereich befinden, drastisch zu senken.

Die Chancen dafür stehen nicht schlecht, denn nachdem CP/M im professionellen Bereich inzwischen durch MS-DOS, ein 16-Bit-Betriebssystem, abgelöst worden ist, sollten die CP/M-Hersteller eigentlich schon an neuen Märkten interessiert sein, und diese neuen Märkte könnten durchaus im oberen Heim-Bereich liegen. Wenn die Software jedoch teurer als der Computer ist, dürfte wohl kein Geschäft zu machen sein.

Fazit

Der C 128 ist ein Gerät einer neuen Leistungsklasse im 8-Bit-Bereich zwischen Homecomputer und Personal Computer. Ganz sicher ist er, bei einem Preis von immerhin knapp 1000 Mark, zumindest auf mittlere Sicht keine Ablösung für den C 64, der für die Hälfte zu haben ist.

Der C 128 ist aber die ideale Maschine sowohl für den »Aufsteiger«, der vom C 64 kommt als auch für den Einsteiger, der einen Computer nicht (nur) zum Spielen sucht, sondern ernsthaft damit arbeiten will.

Dank des CP/M-Betriebssystems ist aus dem Stand jede Menge professioneller Software aus allen Bereichen verfügbar und natürlich auch jede Menge Programmiersprachen. Unter CP/M läuft ja fast alles, was es auch auf Großrechenanlagen gibt, von Cobol und Fortran bis hin zu ausgefeilten Datenbanksystemen.

Und schließlich ist da auch noch das Riesenangebot an C 64-Software, die zwar auch so schon läuft, die aber durch vielfach nur kleine Änderungen auch für den C 128-Modus verfügbar gemacht werden kann. Die meisten Textverarbeitungs-Programme für den C 64 sind mittlerweile auch für den C 128-Modus mit 80-Zeichen am Bildschirm erhältlich oder werden es bald sein.

Fest steht, daß der Markt der Heim- und Personal Computer mit dem C 128 um einen Computer bereichert worden ist, der das Zeug zu einem neuen Stern am 8-Bit-Computerhimmel hat. (ev)

Basic 7.0 – das starke Basic des C 128

Wer auf dem C128 in Basic programmiert, wird angenehm überrascht sein. Mit den ausgezeichneten Befehlen kann der Anwender die Fähigkeiten des Computers voll ausnützen.

Betrachtet man die Entwicklung der Commodore-Computer, so standen sich bisher auf getrennten Seiten stets die Homecomputer und die professionellen Systeme gegenüber. Der C128 schließt nun zum erstenmal die dazwischen klaffende Lücke.

Der Commodore 128 Personal Computer ist ein Computer, der sowohl von seiner Leistungsfähigkeit als auch von seinem ergonomischen Design allen Anforderungen eines professionellen Anwenders genügt.

Basic ist effektiver im C 128-Modus

Nicht nur das neue Styling rückt den C128 näher an die Profis heran, sondern vor allem zwei der drei möglichen Betriebsarten: der C128-Modus und der CP/M-Modus. Im C128-Modus kann man durch Basic effektiver programmieren, der Wortschatz des Basic 7.0 ist der bisher größte Wortschatz eines Commodore-Computers. Im CP/M-Modus steht die CP/M-Version 3.0 zur Verfügung – eine riesige Zahl von professioneller Software hierzu ist bereits auf dem Markt.

Die drei Betriebsarten werden durch zwei Prozessoren erzeugt: Ein MOS 8502-Prozessor ist für den C128-Modus verantwortlich. Außerdem simuliert er den MOS 6510, will man im C64-Modus arbeiten. Der zweite Prozessor ist ein Z80A von Zilog.

Der Grundwortschatz aller Basic-Versionen war das Basic 2.0 des VC20 und C64. Dieses Basic unterstützte allerdings nicht die hervorragenden Fähigkeiten des C64 – wie Grafik, Sprites, Musik. Ein Musikstück zu programmieren oder einen Sprite zu definieren, bedeutet mühsame Kleinarbeit mit PEEK- und POKE-Befehlen. Dies ist sehr zeitraubend und fehleranfällig.

Basic bietet mehr für Grafik, Sprites und Musik

Deshalb hat man für die nächste Generation von Homecomputern, C16, C116 und Plus4, entsprechende Befehle in den Basic-Wortschatz integriert. Zusätzlich sind neue Floppy-Befehle im Basic 3,5 des C16, C116 und Plus4 enthalten. Wie Tabelle 1 zeigt, enthält das Basic 7.0 alle Befehle der vorausgehenden Basic-Versionen. Zusätzlich sind weitere Musik- und Hilfsbefehle enthalten. Vor allem aber ist die Ergänzung von Sprite-Befehlen sehr wichtig. Während der C16, C116, und Plus4 keine Sprites verarbeiten kann, sondern statt dessen »Shapes« besitzt – dies sind kleine bewegliche Figuren, die mit Hilfe von Grafik-Befehlen gezeichnet werden – arbeitet der C128 wieder mit Sprites, die vom Basic kräftig unterstützt werden.

Der C128 ist mit externem Speicher auf 256 KB und 512 KB erweiterbar. Man spricht hierbei von einer RAM-Disk. Für die Arbeit mit der RAM-Disk stehen spezielle Befehle zur Verfügung.

Insgesamt unterscheidet sich der Wortschatz des Basic 7.0 vom Urgroßvater Basic 2.0 durch über 90 zusätzliche Befehle, Anweisungen, Funktionen und Variablen. Im folgenden sind alle diese kurz aufgeführt und erläutert. Bei einigen Befehlen ist die entsprechende Programmierung in Basic 2.0 gegenübergestellt, um die gravierenden Verbesserungen schwerpunktmäßig aufzuzeigen.

(Renate Benz-Heinbücher/zu).

Floppy-Befehle

APPEND:

Eine bereits auf Diskette existierende sequentielle Datei (SEQ und USR) kann hiermit wieder eröffnet und weiter beschrieben werden. Die neuen Daten werden dabei an das Dateiende angefügt. Beim C64 konnte man eine sequentielle Datei nach dem Schließen mit CLOSE nicht weiter beschreiben. Man muß hierzu erst eine Hilfsdatei eröffnen, auf die man alle Daten der alten Datei herüberholt und die neuen Daten dazuschreibt.

C128-MODUS	C64-MODUS	CP/M-Modus
Basic 7.0	Basic 2.0	CP/M V 3.0
128 KB RAM	64 KB RAM	128 KB RAM
48 KB ROM	20 KB ROM	
512 KB-Speicher extern möglich		512 KB-Speicher extern möglich
CPU MOS 8502	MOS 8502 simuliert MOS 6510	Z80A
Taktfrequenz wahlweise 1 MHz oder 2 MHz	Taktfrequenz 1 MHz	Taktfrequenz wahlweise 4 MHz
wahlweise 40 oder 80 Zeichen/ Zeile	40 Zeichen/Zeile	wahlweise 40 oder 80 Zeichen/Zeile
wahlweise 320x200 oder 640x200 Grafikpunkte	320x200 Grafikpunkte	wahlweise 320x200 oder 640x200 Grafikpunkte
16 Farben	16 Farben	8 Farben
8 Sprites	8 Sprites	

Tabelle 1. Die wichtigsten Daten der drei Betriebsarten auf einen Blick:

BACKUP:

Dupliziert eine Diskette auf einem Doppellaufwerk. Hierbei wird die Zieldiskette formatiert. War die Zieldiskette bereits beschrieben, so sind dann die alten Daten alle gelöscht! Verbindet die Befehle HEADER und COPY.

Um eine Diskette mit einem Einzellaufwerk zu kopieren, ist die Zieldis-



kette zu formatieren. Danach ist das zu kopierende Programm erst in den Arbeitsspeicher mit LOAD"name", 8 einzulesen und mit SAVE"name",8 auf die Zieldiskette abzuspeichern. Oder man kann ein Kopierprogramm kaufen. Steht über ein Interface ein Doppellaufwerk für den C 64 zur Verfügung, so kann man dann mit BACKUP die Quelldiskette kopieren. Ist aber nur ein Einzellaufwerk angeschlossen, so muß das zu kopierende Programm erst in den Arbeitsspeicher mit LOAD"name",8 eingelesen und mit SAVE"name",8 auf die Zieldiskette abgespeichert werden.

BLOAD:
Ein Binärfile (das heißt eine Datei mit binär codierten Werten wie Maschinensprache, Sprites usw.) wird von Diskette in den Speicher geladen. Dabei ist die Speicherbank anzugeben, in die es eingelesen werden soll. Beim C 64 nicht möglich.

BOOT:
Ein Binärfile wird von Diskette geladen und bei seiner Startadresse gestartet. Fügt man dem Befehl BOOT keine Parameter an, so sieht das Betriebssystem in Spur 1, Sektor 0 der Diskette in der Floppy mit Gerätenummer 8 nach, was geladen und gestartet werden soll. Diese Information kann man mit Direktzugriffsbefehlen auf Diskette bringen. Beim C 64 nicht möglich.

BSAVE:
Speichert einen Hauptspeicherbereich (Maschinensprache, Binärdaten) als Binärfälle auf Diskette. Beim C 64 nicht möglich.

CATALOG, DIRECTORY:
Beide Befehle sind identisch und geben den Disketteninhalt auf dem Bildschirm wieder. Der Befehl löscht hierbei nicht das im Arbeitsspeicher befindliche Programm! Beim C 64 ist der Disketten-Inhalt abzurufen mit: LOAD"$",8 und auf dem Bildschirm zu zeigen mit LIST. Dabei wird das im Arbeitsspeicher befindliche Programm gelöscht.

COLLECT:
Löscht alle offenen Dateien und gibt den entsprechenden Speicherplatz wieder frei.

CONCAT:
Hängt eine vom Anwender zu bestimmende sequentielle Datei an das Ende einer anderen sequentiellen Datei auf Diskette an.

COPY:
Kopiert eine angegebene Datei unter gleichem oder neuem Namen auf die Diskette im angegebenen Laufwerk einer Doppelfloppy. Arbeitet man innerhalb des gleichen Laufwerks oder mit einem Einzellaufwerk, so dupliziert dieser Befehl die angegebene Datei, doch ist ein neuer Name anzugeben.
Syntax:
COPY (Dquelle)quelldatei TO (Dziel)zieldatei (ON Ugeräteadr).
Beim C 64 ist das Kopieren innerhalb einer Diskette ebenso möglich:
Syntax: OPEN 1,8,15
PRINT#1,"C:neu=alt"
CLOSE1

DCLEAR:
Schließt alle offenen Floppy-Kanäle, nicht die Dateien. Geöffnete Dateien sind vorher mit DCLOSE zu schließen.

DCLOSE:
Schließt eine angegebene oder alle offenen Dateien auf der angeschlossenen Floppy.
Beim C 64: CLOSE n.

DLOAD:
Lädt ein Basic-Programm von Diskette in den Arbeitsspeicher. Bei angeschlossenem Doppellaufwerk ist das Laufwerk anzugeben, auf das zugegriffen werden soll.

DOPEN:
Öffnet eine sequentielle oder relative Datei auf der Diskette für Datenaustausch.

DSAVE:
Speichert ein Basic-Programm vom Arbeitsspeicher auf Diskette. Bei angeschlossener Doppelfloppy ist das gewünschte Laufwerk anzugeben.

DS,DS$:
Diese beiden Systemvariablen liefern den Fehlerstatus der Floppy als Code (DS) und als komplette Statusmeldung (DS$) des zuletzt angesprochenen Floppy-Gerätes. Die Statusmeldung beinhaltet den Fehlercode, die Meldung in Worten und Spur und Sektor, bei denen der Fehler auftrat.
Syntax: PRINT DS, DS$.
Beim C 64: Fehlerabfrage über Fehlerkanal:

```
10 OPEN 1,8,15
20 INPUT#1, A,B$,C,D
30 PRINT A,B$,C,D
40 CLOSE1
```

DVERIFY:
Vergleicht das Programm im Arbeitsspeicher mit einem Programm auf Diskette.
Beim C 64: VERIFY"name",8.

HEADER:
Formatiert eine neue Diskette oder löscht das Inhaltsverzeichnis einer bereits bespielten Diskette.
Syntax: HEADER"name", Dlw (,I id) (,Ugerät).
Beim C 64: OPEN 1,8,15
PRINT#1,"N:name,id"
CLOSE1

RECORD:
Setzt den Schreib/Lese-Zeiger auf eine bestimmbare Position innerhalb einer Relativ-Datei auf Diskette.

RENAME:
Benennt eine Datei auf Diskette um.
Syntax: RENAME alt TO neu (;Dlw) (,Ugerät).

SCRATCH:
Löscht die angegebene Datei aus dem Disketten-Inhaltsverzeichnis. Damit ist die Datei auf Diskette gelöscht.
Syntax: SCRATCH name (,Dlw) (,Ugerät).

Die Befehle HEADER und SCRATCH haben, um versehentliches Löschen zu vermeiden, die Sicherungsrückfrage ARE YOU SHURE? eingebaut. Erst nach Bejahen dieser Frage mit Y oder Yes wird formatiert, beziehungsweise gelöscht.

Programmierhilfen
Befehle für RAM-Disk,
Maschinensprache-
Befehle (Farb-, Print-Befehle,
Variablen, Funktionen etc.)

AUTO:
Automatisiert die Zeilennumerierung mit angegebener Schrittweite. Nach AUTO wird die folgende Zeilennummer angezeigt, wenn man am Ende einer Basic-Zeile RETURN drückt.

BANK:
Dieser Maschinensprachebefehl legt eine 64KB Speicherbank für PEEK-, POKE-, SYS- oder WAIT-Anweisungen fest. Syntax: Bank n (n zw. 0 u. 15).

BEGIN...BEND:
Stehen immer im Zusammenhang mit der IF...THEN-Anweisung. Im Gegensatz zu IF...THEN...ELSE (siehe später) schließen BEGIN... BEND, nach THEN stehend, mehrere Basic-Anweisungen ein, die sich auch über mehrere Zeilen erstrecken können.
Beispiel:

```
5 GETA$:IFA$=""THEN5
10 IF A$="J" THEN BEGIN: X=1
20 Y=2: Z=3
30 PRINT X*Y*Z:
   BEND: PRINT"=X*Y*Z"
40 PRINT"WEITER MIT SPACE"
50 .....
```

Die Tastatur wird mit GETA$ abgefragt. Ist der Buchstabe J gedrückt worden, so trifft das Programm auf BEGIN und arbeitet alle danach folgenden Anweisungen

bis BEND ab, fährt nach BEND im Programm fort. Wird irgendeine andere Taste gedrückt, so ist A$="J" logisch falsch und das Programm wird in Zeile 20 fortgesetzt. Trifft es auf BEND, so kann es diese Anweisung nicht verstehen, da es die Anweisung BEGIN übersprungen hatte. Das BEND ist dann eine Aufforderung, in der nächsten Programmzeile fortzufahren. Das heißt, die Anweisung PRINT"=X*Y*Z" wird nur ausgeführt, wenn A$="J", also die Taste J gedrückt wurde.

CHAR:

Schreibt einen String (Zeichenkette) an anzugebender Position auf den Bildschirm. Diese Anweisung arbeitet im Text- und Grafik-Modus.

COLOR:

Legt für die Farbquellen
- 0 Hintergrund (40 Z-Darstellung)
- 1 graf. Vordergrund (bei HIRES)
- 2 Zusatzfarbe 1 bei Multicolor-Grafik
- 3 Zusatzfarbe 2 bei Mulitcolor-Grafik
- 4 Rand
- 5 Textfarbe
- 6 Hintergrund (80 Z-Darstellung)

eine der möglichen 16 Farben fest.
Farbcodes: 1 schwarz, 2 weiß, 3 rot, 4 grün, 5 violett, 6 dunkelgrün, 7 blau, 8 gelb, 9 hellbraun, 10 braun, 11 rosa, 12 dunkelgrau, 13 grau, 14 hellgrün, 15 hellblau, 16 hellgrau.

Beispiel:
COLOR0, 2:COLOR1, 3:COLOR4,1.
Der Hintergrund erscheint weiß, der Vordergrund (Text) rot und der Rahmen schwarz. Beim C64 wird die Hintergrund- und Rahmenfarbe mit POKE-Befehlen festgelegt: POKE53281,2:POKE53280,1. Die Farbe des Textes wird in der PRINT-Anweisung definiert.

DELETE:

Löscht den durch die Zeilennummern zu bestimmenden Bereich des Programms im Arbeitsspeicher.

DO...(UNTIL)..(EXIT)..LOOP..(UNTIL)..bzw.
DO...(WHILE)..(EXIT)..LOOP..(WHILE):

Die Anweisungen DO...LOOP definieren eine Programmschleife, die unendlich oft durchlaufen wird. Wie FOR...NEXT-Schleifen können auch diese geschachtelt sein. Um eine solche DO...LOOP-Schleife zu verlassen, gibt es 3 Möglichkeiten:
1. EXIT: Durch diese Anweisung springt das Programm an der entsprechenden Stelle, an der sich das EXIT befindet, aus der Schleife und setzt das Programm hinter der nach LOOP folgenden Anweisung fort.
2. UNTIL: Das Programm durchläuft die Schleife so lange, bis (englisch:until) der nach UNTIL folgende Ausdruck logisch wahr wird.
3. WHILE: Das Programm durchläuft die Schleife solange der nach WHILE folgende Ausdruck wahr ist (das heißt, bis er falsch wird).

FAST:

Schaltet den Prozessor MOS 8502 von der Taktfrequenz 1 MHz auf die doppelt so schnelle Arbeitsweise im 2 MHz-Takt um. Beim 2 MHz-Betrieb ist die Bildschirm-Anzeige bei der 40 Zeichen-Darstellung abgeschaltet. Der für die 40 Zeichen-Darstellung verantwortliche VIC-Chip ist der gleiche wie im C64. Er kann die schnelle Taktfrequenz nicht mithalten. Für die 80 Zeichen-Darstellung ist ein anderer VIC-Chip zuständig, der schneller arbeitet. Deshalb bleibt die Bildschirm-Anzeige dabei erhalten.

FETCH:

Holt, das heißt bringt eine anzugebende Byte-Menge von der RAM-Disk (Speichererweiterungsbank) in den Arbeitsspeicher.
Beispiel:
FETCH 1000,52000,7,2000 überträgt 1000 Byte aus Speicherbank 7 ab Adresse 2000 in den BASIC-Arbeitsspeicher ab Adresse 52000.

GETKEY:

Vergleichbar mit GET (GET liest Tastatur, ohne zu warten). Diese Anweisung wartet so lange, bis eine Taste gedrückt wird. Diese nur im Programm-Modus anwendbare Anweisung überträgt den ASCII-Code der gedrückten Taste in die angegebene Variable.

G064:

Schaltet den C128 vom C128-Modus in den C64-Modus um. Im Direkt- und Programm-Modus einsetzbar!

HELP:

Bringt – wie die HELP-Taste – nach einer Fehlermeldung die fehlerhafte Programmzeile auf dem Bildschirm und zeigt den fehlerhaften Teil unterlegt (40Z9 beziehungsweise unterstrichen (80Z) an.

IF...THEN...ELSE:

Verzweigt in verschiedene Programmteile, je nach dem logischen Wahrheitsgehalt eines dem IF folgenden Ausdrucks. Ist der nach IF folgende Ausdruck wahr, so arbeitet das Programm nach THEN weiter. Ist dieser Ausdruck falsch, so springt das Programm, wenn kein ELSE vorhanden ist, in die nächste Zeile. Ist aber das ELSE vorhanden, so werden die danach folgenden Anweisungen ausgeführt. THEN und ELSE müsssen in der gleichen Zeile stehen. Nur eine BEGIN-Anweisung (siehe weiter vorn) kann sich über mehrere Zeilen erstrecken.

KEY:

Dieser Befehl bezieht sich nur auf die Funktionstasten. Er fragt die Belegung der Funktionstasten ab oder belegt sie neu. Beispiel:
a) momentane Belegung wird mit KEY ohne Parameter angezeigt.
b) KEY1,"COLOR": F1 schreibt das Wort COLOR auf den Bildschirm. Man gibt dann die gewünschten Parameter ein und erhält nach RETURN die entsprechende Farbgebung.

MONITOR:

Ruft den Maschinensprache-Monitor auf.

PRINT USING:

Formatiert die Ausgabe auf Bildschirm oder Drucker, wobei das Druckformat als Steuerzeichen in einem String abzulegen ist.
Syntax:
PRINT (#filenr.) USING v$; liste (;)
wobei mit v$ das Druckformat definiert wird (Tabelle im Handbuch).
Beispiel:
PRINT USING ".";"23" liefert das Ergebnis 23.00.

PUDEF:

Definiert bis zu vier unterschiedliche Steuerzeichen neu, die in der PRINT-USING-Anweisung bereits festgelegt sind.

RENUMBER:

Numeriert das im Arbeitsspeicher stehende BASIC-Programm ganz oder abschnittweise neu. Anzugeben ist hierbei die Zeilennummer, mit der die neue Numerierung beginnen soll (Default ist 10), die Schrittweite der neuen Numerierung (Default ist 10) und die Zeilennummer im Programm, ab der neu zu numerieren ist (Default ist 10).

RESUME:

Am Ende eines Unterprogramms zur Fehlerbehandlung (siehe TRAP) setzt diese Anweisung das Programm mit der nach RESUME folgenden Zeilennummer fort.
Beispiel:

```
1000   TRAP 20000
 .
 .
20000  IF ER=74 THEN
       RESUME 30000
 .
```



30000 PRINT"FLOPPY ÜBERPRÜFEN"

Bei dem Fehler DRIVE NOT READY (Fehlercode 74) erscheint für den Anwender die Aufforderung auf dem Bildschirm, daß er die Floppy überprüfen soll. Zum Beispiel kann es sein, daß sich keine Diskette im Laufwerk befindet, aber ein Zugriff stattfinden sollte.

RREG:

Liest die Prozessorregister A,X,Y und SR in die vom Anwender angegebenen Variablen (am Ende eines mit SYS aufgerufenen Unterprogramms in Maschinensprache).

RUN:

Ergänzt Anwendung des RUN-Befehls:

RUN"programmname" lädt und startet das spezifizierte Programm.

SCNCLR:

Löscht den vom Anwender anzugebenden Bildschirm. Syntax: SCNCLR (n), wobei n=0,...5 dem Grafik-Modus entspricht.

SLEEP:

Hält die Programmausführung für eine bestimmbare Zeit an. Ersetzt eine Warteschleife. Syntax: SLEEP n, mit n zw. 1 und 65535.

SLOW:

Schaltet den Prozessor von der 2 MHz-Taktfrequenz wieder auf 1 MHz um. Der Rechner ist dann nur noch halb so schnell, aber die Bildschirmanzeige bei der 40-Zeichen-Darstellung ist wieder aktiviert.

STASH:

Eine weitere Anweisung für die Arbeit mit der RAM-Disk. Sie arbeitet in umgekehrter Richtung wie FETCH: eine vom Anwender anzugebende Byte-Menge geht vom Basic-Arbeitsspeicher in die RAM-Disk.

Beispiel:

STASH 1000,520000,7,2000 überträgt 1000 Byte ab Adresse 52000 aus dem Basic-Arbeitsspeicher in die Speicherbank 7 ab Adresse 2000.

SWAP:

Überträgt eine anzugebende Menge von Bytes aus dem Arbeitsspeicher auf die RAM-Disk und gleichzeitig wird eine gleich große Anzahl von Bytes aus dieser Stelle der RAM-Disk an die frei gewordene Adresse im Arbeitsspeicher übertragen. Dies entspricht einem Austausch von Bytes.

TRAP:

Bei einem Fehler findet normalerweise eine Programmunterbrechung statt. Die Anweisung TRAP führt jedoch dazu, daß das Programm bei einem Fehler nicht unterbrochen wird, sondern zu der nach TRAP folgenden Zeilennummer verzweigt. Dies unterdrückt eine Fehlermeldung und ermöglicht eine Fehlerbehandlung in einem Unterprogramm. Syntax: TRAP n, wobei die n die Zeilennummer ist, mit der die Fehlerbehandlungsroutine beginnt. Mit RESUME wird dann das Hauptprogramm fortgesetzt.

TRON/TROFF:

Schaltet die Programmablaufverfolgung ein oder aus (trace on beziehungsweise trace off). Dies bedeutet, daß beim Austesten eines Basic-Programmes die aktuelle Zeilennummer schrittweise angezeigt wird.

WINDOW:

Definiert ein Bildschirmfenster durch Angabe der Koordinaten seiner linken oberen und rechten unteren Ecke oder löscht das Fenster. Syntax: WINDOW xlo, ylo, xru, yru (,lö) wobei lö=0 oder 1, je ob löschen (1) oder nicht (0), Default ist 0.

Systemvariablen

EL,ER:

EL liefert die Nummer der Programmzeile, in der ein Fehler aufgetreten ist.

Beispiel:

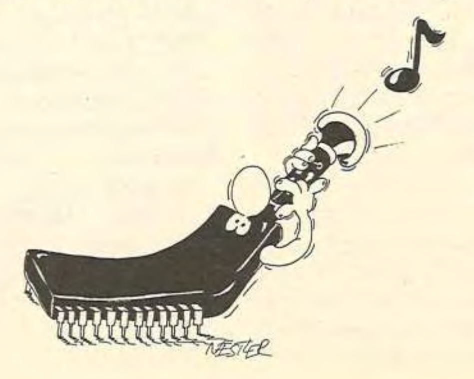
100 IF EL=100 THEN 1000.

ER liefert den Code des zuletzt vom Interpreter diagnostizierten Fehlers.

Beispiel:

100 IF ER=30 THEN 1000.

Wenn die Stop-Taste gedrückt wird, verzweigt das Programm nach Zeile 1000. 30 ist der Fehlercode für einen BREAK.



Funktionen

DEC, ERR$, HEX$, INSTR, JOY, PEN, POINTER, POT, RCLR, RGR, RWINDOW, XOR:

Für Programmierung in Maschinensprache wichtig: DEC gibt den Dezimalwert eines Zahlenwertes in hexadezimaler Schreibweise an.

Beispiel: PRINT DEC("FFFF") liefert das Ergebnis 65535.

HEX$ gibt umgekehrt die hexadezimale Darstellung eines Dezimalwertes n an. (n zwischen 0 und 65535).

Beispiel:

PRINT HEX$(1456) liefert 5B0.

ERR$ gibt die Fehlermeldung in Worten wieder, entsprechend dem Fehlercode n (n zwischen 1 und 127).

Beispiel:

PRINT ERR$(ER).

INSTR gibt die Position eines vom Anwender anzugebenden Teilstrings in einem String an (string=Zeichenkette).

Beispiel:

PRINT INSTR("COMMODORE", "DO")

liefert das Ergebnis 6.

JOY gibt einen Wert für die Position des Joysticks und den Zustand des Feuerknopfes an. Die den Werten entsprechenden Zustände sind dem Handbuch zu entnehmen.

Beispiel:

PRINT JOY(2) liefert 135: Joystick 2 steht bei gedrücktem Feuerknopf in der Stellung nach links.

PEN gibt die Bildschirmkoordinaten des Lichtstiftes an.

POINTER (für Maschinensprache wichtig) gibt die Speicheradresse des ersten Bytes eines Variablenzeigers in der Speicherbank 1 an. In dieser Speicherbank sind vom Interpreter die Variablen Strings und Felder abgelegt.

POT gibt die Position eines der 4 anschließbaren Drehregler an.

RCLR gibt den Farbcode (1 – 16) für die vom Anwender angegebene Farbquelle an. (siehe COLOR weiter vorn).

RGR gibt den momentanen Grafik-Modus an (0 – 5, siehe GRAPHIC).

RWINDOW gibt die Koordinaten des momentanen Bildschirmfensters an.

XOR liefert das Ergebnis einer Exklusiv-Oder-Verknüpfung zweier ganzer, vorzeichenloser 16-Bit-Werte. Beispiel: PRINT XOR (124,12); XOR(0,12) liefert als Ergebnis 112 12.

Grafik-Befehle:

BOX:

Zeichnet ein Rechteck an angegebener Stelle auf dem Bildschirm und kann dieses auch ausmalen.

Syntax:

BOX (farbquelle), x1, y(, (x2,y2)) (,(winkel)) (,(ausmal)), wobei Farbquelle=0 Hintergrund bedeutet,
1 Vordergrund,
2 Zusatzfarbe 1,
3 Zusatzfarbe 2.
x1, y1 Koordinaten der linken oberen Ecke, x2,y2 Koordinaten der rechten unteren Ecke. x-Werte liegen zwischen 0 und 320 im HIRES-Modus, zwischen 0 und 160 im Multicolor-Modus, y-Werte liegen zwischen 0 und 199. Der Ursprung liegt in der linken oberen Bildschirmecke. Der Winkel (Grad) gibt die Drehung des Rechtecks um seinen Mittelpunkt an. Default=0. Der grafische Cursor (auch Pixel-Cursor genannt) befindet sich am Ende des Zeichnens in der rechten unteren Ecke x2,y2.

CIRCLE:

Zeichnet einen Kreis, eine Ellipse oder ein beliebiges Polygon mit dem Mittelpunkt x,y auf dem Bildschirm. Anzugeben ist die Farbquelle, die Mittelpunktskoordinaten, Radius der Hauptachse, Radius der Nebenachse, Winkelangabe für Drehung der Figur, Segmentwinkel.

DRAW:

Verbindet die angegebenen Koordinaten durch eine Linie.

Syntax:

DRAW (farbquelle),(x1,y1) TO x2,y2. Hierbei sind x1,y1 die Koordinaten des Anfangspunktes und x2,y2 die Koordinaten des Endpunktes der Linie. Nur die Länge einer Programmzeile beschränkt die Anzahl der Koordinaten, die durch TO miteinander verbunden werden. Stimmen die Koordinaten x1,y1 und x2,y2 überein, so wird ein Punkt gezeichnet.

GRAPHIC:

Ruft einen der 6 verfügbaren Grafik-Modi auf und belegt den Grafik-Speicher (bit-map) beziehungsweise gibt diesen wieder frei.

Syntax: GRAPHIC modus (,(lösch), textz).

modus:

0 = Text mit 40 Zeichen (default)
1 = hochauflösende Grafik (320x200 Punkte)
2 = hochauflösende Grafik, geteilter Bildschirm
3 = Mehrfarbengrafik (160x200 Punkte)
4 = Mehrfarbengrafik, geteilter Bildschirm
5 = Text mit 80 Zeichen

lösch: O BS nicht löschen, 1 BS löschen.

textz: 0–24, legt für den Modus 2 oder 4 fest, in welcher Zeile der Text beginnen soll (Default ist 19). GRAPHIC CLR gibt reservierten Speicher wieder frei.

LOCATE:

Setzt den nicht sichtbaren Pixel-Cursor an gewünschte Stelle auf den Bildschirm.

PAINT:

Füllt eine geschlossene Fläche mit der angegebenen Farbe aus. Koordinaten innerhalb der auszumalenden Fläche sind anzugeben.

SCALE:

Mit dieser Anweisung kann man die Skalierung der Bildschirmkoordinaten verändern. Er schaltet ebenso die Skalierung ein oder aus. Die Grenzen für die neue Skalierung: der maximale x-Wert muß im HIRES-Modus zwischen 320 und 1023, im Mehrfarben-Modus zwischen 160 und 1023 liegen, der maximale y-Wert zwischen 200 und 1023.

Syntax:

SCALE (n)(,xmax, ymax) wobei n=0 die Skalierung aus- und n=1 die Skalierung eingeschaltet.

WIDTH:

Legt die Strichstärke für alle Zeichenanweisungen (wie BOX, CIRCLE...) fest. Einfache oder doppelte Strichstärke ist möglich. Syntax: WIDTH n, mit n=1 oder n=2.

Funktionen

RDOT:

Gibt die momentane Bildschirm-Position des Pixel-Cursors an oder den entsprechenden Farbquellcode.

Syntax:

RDOT(n)m n=0: x-Position, n=1: y-Position, n=2: Farbquellcode.

Sprite-Befehle

Sprites sind selbst entworfene Figuren, die auf dem Bildschirm bewegt werden können. Sie bestehen aus einer 24x21 Punktmatrix, beziehungsweise aus einer 12x21 Punktmatrix im Mehrfarbenmodus. Der C 128 kann wie der C 64 acht solcher Sprites gleichzeitig auf dem Bildschirm bewegen.

Im C 128-Modus kann man auf 3 Arten Sprites erzeugen:

1. Mit POKE-Anweisungen wie beim C 64. Hierbei wird das Bitmuster der Sprite-Matrix in spezielle Speicherplätze gePOKEt.
2. Mit dem SPRDEF-Befehl.
3. Mit den SSHAPE-, SPRSAV- und SPRITE-Anweisungen.

Zu 2) Der SPRDEF-Befehl ist der schnellste und einfachste Weg, Sprites zu erzeugen. Er schaltet den Sprite-Editor ein. Mit Hilfe dieses Editors kann ein Sprite direkt in einer auf dem Bildschirm erscheinenden Punktematrix definiert werden. Geht man aus dem Editor wieder heraus, so können Sie die Sprite-Werte mit BSAVE auf Diskette speichern oder den Sprite mit SPRITE aktivieren und mit MOVSPR bewegen.

Zu 3) a. Man zeichnet ein beliebiges Bild mit den Befehlen DRAW-, CIRCLE, BOX und PAINT. Die Größe richtet sich nach der 24x21 beziehungsweise 12x21 Punktematrix.

b. Die SSHAPE-Anweisung speichert dieses Bild in einer Zeichenkettenvariablen.

c. Dann überträgt die Anweisung SPRSAV diesen Entwurf in die Sprite-Grafik.

d. Die Anweisung SPRITE aktiviert diesen SPRITE, weist ihm eine Farbe zu, vergrößert ihn.

e. Mit MOVSPR kann man ihn bewegen.

Die Sprite-Befehle genau unter die Lupe genommen

COLLISION:

Aktiviert eine Programmunterbrechung für Sprite-Kollision. Kollidieren zwei Sprites oder ein Sprite mit der Bildschirmanzeige, so kann man mit dieser Anweisung in ein Unterprogramm verzweigen. In diesem Unterprogramm kann die Kollision zum Beispiel akustisch verarbeitet sein. Mit dieser Anweisung ist die Programmunterbrechung aber auch zu inaktivieren.

Syntax:

COLLISION typ (,zeilennummer) wobei
typ =1: Sprite-Sprite-Kollision,
typ =2 Sprite-Anzeige-Kollision,
typ =3 Lichtstift-Aktivierung zeilennummer muß eine vorhandene Programmzeilennummer sein, mit der bei Unterbrechung das Programm weiter fortfährt.

GSHAPE:

Bringt den Wert eines Strings als binäre Bildinformation auf den Grafik-Bildschirm.

MOVSPR:

Bewegt ein Sprite mit einer angegebenen Geschwindigkeit und in gegebener Richtung. Ebenso setzt dieser Befehl einen Sprite an die gewünschte Position auf dem Bildschirm.

SPRCOLOR:



Definiert im Mehrfarben-Modus die Zusatzfarben.
SPRDEF:
Ruft den Sprite-Editor auf. Es erscheint ein Raster von 24x21 Punkten. In diesem Raster erzeugt der Anwender seinen Sprite indem er jeweils einen Punkt setzt oder nicht. Die Farbe des Sprites ist mit den Farbtasten der Tastatur zu bestimmen, die Ausdehnung in x- und / oder y-Richtung ist ebenfalls ganz einfach mit den Tasten x und y zu erreichen und vieles mehr. Mit BSAVE kann man darauf diesen Sprite oder diese Sprites auf Diskette speichern und mit BLOAD wieder einladen.
SPRITE:
Legt die folgenden Eigenschaften für einen Sprite fest: Farbe, Priorität, Dehnung, Modus.
Beispiel: SPRITE3,1 aktiviert (zweite 1) das Sprite Nummer 3, das heißt Sprite 3 ist dann auf dem Bildschirm zu sehen.
SPRSAV:
Überträgt den Sprite-Entwurf (mit Hilfe von Zeichenanweisungen), das heißt die String-Variable an die Sprite-Grafik. Aber auch umgekehrt speichert er ein bestimmtes Sprite als Punkteraster in eine Stringvariable.
SSHAPE:
Speichert den Inhalt eines angegebenen Bereichs auf dem Grafikbildschirm (zum Beispiel Entwurf eines Sprites) als binäre Bildinformation in eine Zeichenkettenvariable.
Beispiel:
1000 SSHAPE A$,11,10,34,31
1010 SPRSAV A$,1.
Zeile 1000 speichert das Shape (Rechteck mit den Koordinaten 11,10 für linke obere Ecke und 34,31 für rechte untere Ecke) als binär codierte Information im String A$. Zeile 1010 speichert den Inhalt von A$ in das Sprite 1.

Funktionen

BUMP:
Gibt die Nummer des Sprites (1–8), das seit der letzten BUMP-Abfrage entweder mit einem anderen Sprite (n=1) oder mit der Bildschirmanzeige (n=2) kollidiert ist.
Syntax: BUMP(n).
RSPCOLOR:
Gibt den Farbcode der Zusatzfarbe für Sprites.
RSPPOS:
Gibt für gewünschtes Sprite Bildschirmposition oder momentane Geschwindigkeit an.
Syntax:
v=RSPPOS(n,m), wobei n=1,..,8 das zu untersuchende Sprite definiert und m=0 aktuelle x-Koordinate, m=1 y-Koordinate, m=2 Geschwindigkeit liefert.
RSPRITE:
Gibt für gewünschten Sprite dessen zum Beispiel mit SPRITE definierte Eigenschaften an, wie Farbe, Dehnung etc.

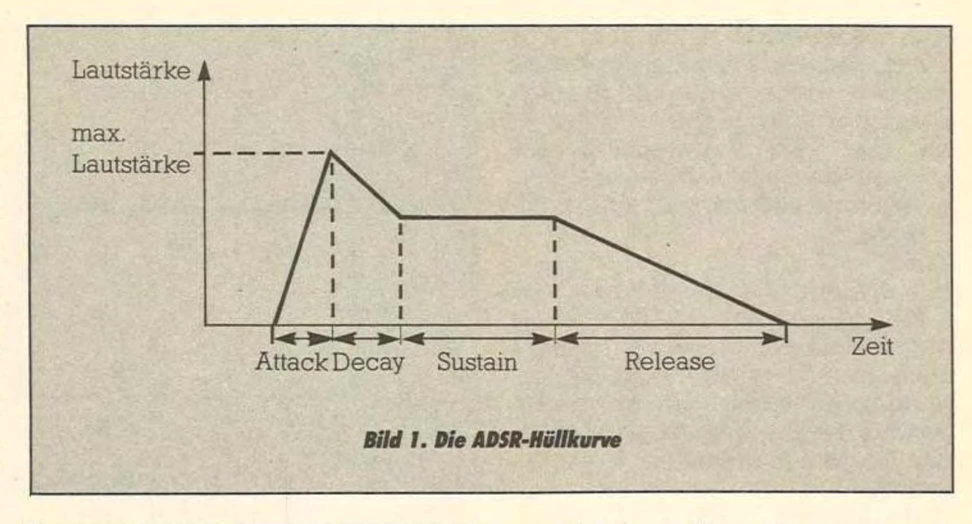


Bild 1. Die ADSR-Hüllkurve



Bild 2. Verschiedene Schwingungsformen

Musik-Befehle

Verschiedene Parameter beeinflussen die Klangfarbe eines Tones: die ADSR-Parameter, die Wellenform, der Filter. Der Klangbaustein des C 128 (SID-Chip) erlaubt eine Programmierung dieser Parameter. Hierfür stehen eine Reihe von Basic-Vokabeln zur Verfügung.
ENVELOPE:
Mit ENVELOPE ändert man die Klangfarbe eines Tones. Dazu benötigt der Befehl Werte, die den Lautstärkeverlauf des Tones wiedergeben, die sogenannten ADSR-Parameter (Attack, Decay, Sustain, Release). Diese vier Werte bestimmen die Anschlagzeit (Attack), die Abschwellzeit (Decay), die Haltezeit (Sustain) und die Ausklingzeit (Release) des Tones (siehe Bild 1).
Diesen Lautstärkenverlauf eines Tones bezeichnet man als Hüllkurve.
Die Klangfarbe eines Tones ist aber auch über die Wellenform beeinflußbar. Der C 128 kann vier verschiedene Tonwellen erzeugen (Bild 2).
Bei ENVELOPE ist eine dieser vier Wellenformen anzugeben.
Der Befehl ENVELOPE kann durch Angabe der Impulsbreite für die Rechteckwelle die Klangfarbe nochmals verändern.
Mit ENVELOPE erstellt man also eigene Hüllkurven und Klangformen (zehn sind möglich). Der C 128 hat aber bereits die Klangfarben für zehn verschiedene Musikinstrumente vorprogrammiert, das heißt die oben aufgezählten Parameter für diese Instrumente sind bereits mit entsprechenden Werten einprogrammiert.
Diese Instrumente sind:
Klavier
Akkordeon
Zirkusorgel (Calliope)
Trommel
Flöte
Gitarre
Cembalo
Orgel
Trompete
Xylophon
Mit ENVELOPE können Sie diese vordefinierten Parameter jedoch abändern.
FILTER:
Der SID-Chip simuliert drei Filtertypen, die die Wellenform hervorheben oder eliminieren:
1. Tiefpaß-Filter
Er läßt niedrige Frequenzen, das heißt die Frequenzen unterhalb der angegebenen Grenzfrequenz durch und sperrt die hohen, oberhalb der Grenzfrequenz liegenden Frequenzen (siehe Bild 3). (Ton dumpf).
2. Hochpaß-Filter
Gerade umgekehrt läßt dieser die hohen Frequenzen durch, während er die unter der Grenzfrequenz liegenden Frequenzen sperrt (siehe Bild 4). (Ton scharf).
3. Bandpaß-Filter

Nur Frequenzen innerhalb des angegebenen Frequenzbereiches kommen durch (siehe Bild 5) . (Ton hohl). Der entsprechende Filter ist mit der FILTER-Anweisung ein- oder auszuschalten. Alle drei Filtermöglichkeiten kann man auch kombinieren.

PLAY:

Mit diesem Befehl können Sie Noten spielen, die in einem String definiert sind. In diesem String sind außer den Noten aber auch Steuerzeichen enthalten, die Anweisungen für die Tonerzeugung oder für das Abspielen enthalten.

Syntax:

PLAY A$ oder PLAY "CDEFGAB", letzteres spielt die Tonleiter, die Note B wird im Deutschen als H bezeichnet. Die folgenden Zeichen kann man – als Steuerzeichen – in den String einbetten, um Änderungen vorzunehmen. Diese Steuerzeichen muß man nicht benutzen, sie nutzen nur die Fähigkeiten des Synthesizers besser aus.

On: bestimmt eine von sieben Oktaven (n=0,..,6),
Tn: legt eine der folgenden zehn Klanghüllkurven fest:
- 0 – Klavier
- 1 – Akkordeon
- 2 – Zirkusorgel
- 3 – Trompete
- 4 – Flöte
- 5 – Gitarre
- 6 – Cembalo
- 7 – Orgel
- 8 – Trompete
- 9 – Xylophon

Un: bestimmt die Lautstärke (n=0,...,9)
Vn: bestimmt eine von drei möglichen Stimmen
Xn: schaltet das mit der FILTER-Anweisung gewählte Filter ein (n=1) oder aus (n=0)
N: eine der Noten A B C D E F G
#N: Note N einen Halbton höher
$N: Note N einen Halbton tiefer
W: Note wird als ganze Note gespielt
H: Note wird als halbe Note gespielt
Q: Note wird als viertel Note gespielt
I: Note wird als achtel Note gespielt
S: Note wird als sechzehntel Note gespielt
.N: Note wird als punktierte Note (die Hälfte ihres Wertes) gespielt
R: setzt Pause
M: wartet, bis die momentan gespielte Musik zu Ende ist.

SOUND:

Ton- und Klangeffekte sind mit SOUND einfach und schnell zu erzeugen. Die SOUND-Anweisung gibt diese dann durch den Lautsprecher des Monitors oder TVs wieder. SOUND benötigt Parameter wie Angabe der Stimme (beziehungsweise Ton), der Frequenz, Dauer des Tones, Richtung (das heißt ob zunehmende, abnehmende oder oszillierende Klangstufe), Maximalfrequenz (für anschwellende Klangeffekte), Stufe (bestimmt Zeitdauer für stufige Klangeffekte), Wellenform, Impulsbreite.

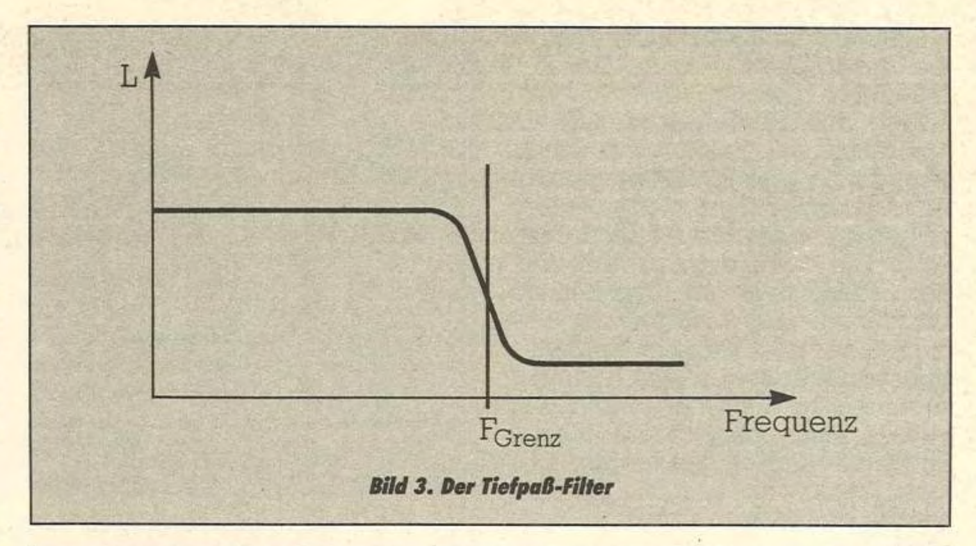


Bild 3. Der Tiefpaß-Filter

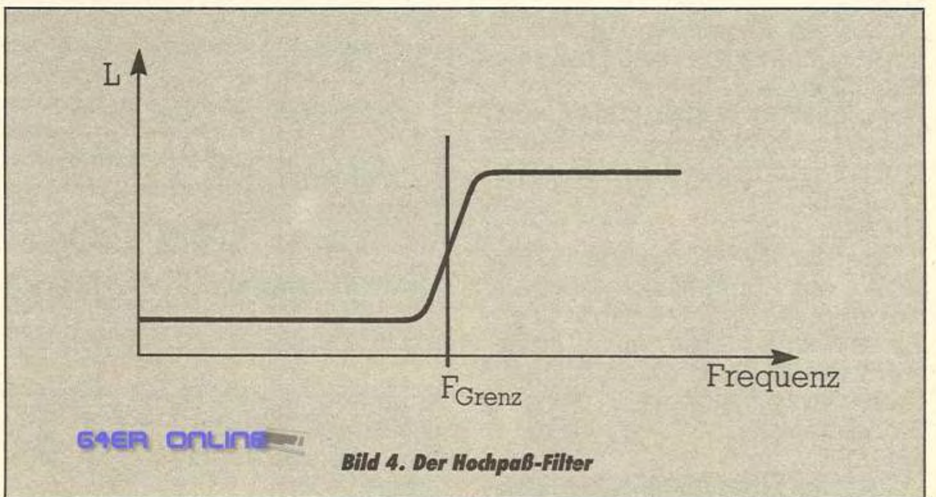


Bild 4. Der Hochpaß-Filter

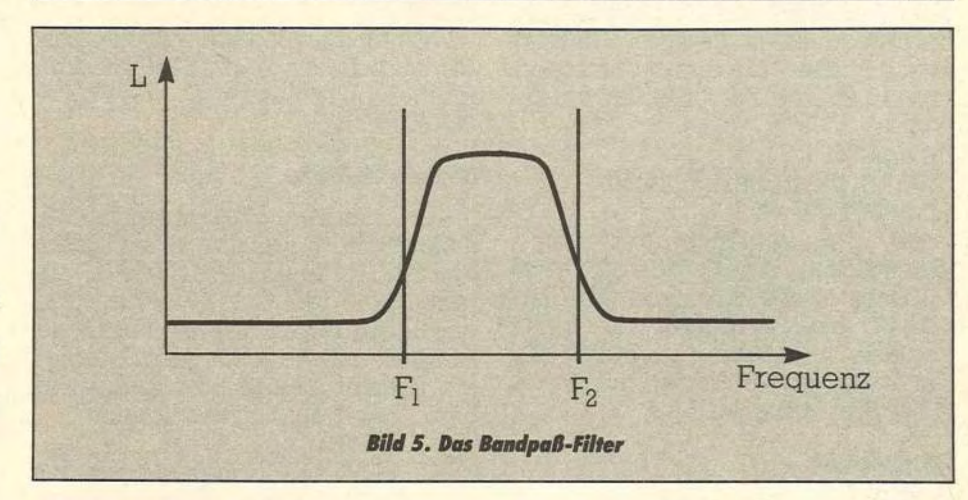


Bild 5. Das Bandpaß-Filter

TEMPO:

Definiert das Spieltempo für die Noten.

Syntax:

TEMPO n, wobei n=0,...,255 die Dauer angibt, die Dauer entspricht 19.22/n Sekunden.

VOL:

Bestimmt die Lautstärke für die mit der SOUND- und der PLAY-Anweisung entwickelten Geräusche beziehungsweise eines Musikstückes. Syntax: VOL n, wobei n=0,...,15.

(zu)



Basic 7.0-Programme mit Struktur

Das neue Basic 7.0 liefert nicht nur tolle Befehle, sondern auch einen Vorteil, wie man ihn sonst nur bei höheren Programmiersprachen findet: Es unterstützt strukturiertes Programmieren.

Die große Basic-Überraschung findet vor allen Dingen bei den Aufsteigern vom Commodore 64 zum C128 statt. Endlich Befehle, mit denen man die Fähigkeiten des Computers auch ausnützen kann.

Diese überragenden Basic-Befehle sieht man nur, wenn man Programme vom Commodore 64 mit demselben Programm im Basic 7.0 des C128 vergleicht. Das sieht dann bei einer Grafik aus wie in Listing 1.

Für die Lissajous-Figur werden im Basic 2.0 des Commodore 64 14 Programmzeilen gebraucht. Dagegen benötigt der Commodore 128 nur sechs Programmzeilen und das Programm sieht wesentlich aufgeräumter aus (Listing 2).

Noch krasser ist der Unterschied, wenn ein kleines Liedchen programmiert wird. Dazu braucht der C64 etliche Datazeilen, einen Wulst von POKEs und Schleifen (Listing 3).

Der Commodore 128 braucht für das gleiche Programm ganze vier Programmzeilen (Listing 4).

Neben den neuen Sound-, Grafik- und Datei-Befehlen sind auch neue Befehle für das komfortablere Programmieren hinzugekommen. In erster Linie sind das Programmierhilfen wie TRON oder HELP, die dem Programmierer das Testen von Basic-Programmen wesentlich erleichtern. Daneben gibt es aber auch eine kleine Gruppe von Befehlen, die die logische Struktur eines Basic 7.0-Programmes enorm vereinfachen.

Kleine Befehlserweiterungen – große Wirkung

Das Commodore Basic 2.0 kennt bereits zwei Befehle, mit denen ein Programm übersichtlich wird: FOR...TO...NEXT sowie IF...THEN. Allerdings waren hiermit keine weitreichenden Aufgaben zu erfüllen, weil die Anschlußbedingungen immer in eine Zeile passen mußten. Die Folge war ein mit GOTO und GOSUB gespicktes Listing. Wer schon ein solches Programm mit vielen Abfragen und Programmverzweigungen programmiert hat, weiß, warum man diese Art von Programmen mit Spaghettis vergleicht. Das Programm »wurstelt« sich von einer Zeile zur nächsten, verzweigt in andere Programmzeilen und springt mehrmals in die gleichen Abfragen. Eine Analyse des Programmes ist je nach Länge sehr schwierig. Ein Beispiel für diese Art von Programmierung ist das Listing »Hamburger« (Listing 5).

Für die Abfrage einer Bedingung wird in Basic meist die Anweisung IF..THEN benutzt. Dieser Befehl ist auf dem Commodore 128 wesentlich verbessert worden. Er erhielt ein Element, wie man es normalerweise nur von höheren Programmiersprachen kennt: die ELSE-Anweisung.

Wenn das Wörtchen ELSE nicht wär

Bei der Verwendung von IF... THEN im C64-Basic muß man fast immer auch mit dem Befehl GOTO oder GOSUB arbeiten, da nur wenige Befehle hinter der Anweisung THEN Platz finden. Hat man mehrere Befehle, muß man in ein eigenes Programmteil springen und nach der Ausführung wieder zurückkehren. Dadurch wird umständlich im Basic-Programm verzweigt beziehungsweise herumgesprungen.

Die erste wesentliche Hilfe für bessere Programmstrukturen ist der Befehl ELSE. Der IF-Befehl sieht dann folgendermaßen aus: IF...THEN...ELSE.

Wenn die IF-Bedingung erfüllt wird, dann wird die hinter THEN stehende Anweisung ausgeführt. Ansonsten wird – sofern eine ELSE-Anweisung vorhanden ist – die ELSE-Bedingung ausgeführt.

```
10 INPUT"WER HAT BASIC 7.0";A$
20 IF A$="COMMODORE 128" THEN
PRINT"RICHTIG":END:ELSE PRINT
"FALSCH":GOTO 10
```

Bei unserem Programmbeispiel wird in Zeile 10 auf eine Eingabe gewartet. In Zeile 20 wird bei richtiger Beantwortung das Programm beendet, im anderen Fall wird weiter gefragt. Der Commodore 64 bräuchte eine Zeile mehr, das Listing sieht bei ihm folgendermaßen aus:

```
10 INPUT"WER HAT BASIC 2.0";A$
20 IF A$="COMMODORE 64" THEN
PRINT"RICHTIG":END
30 PRINT"FALSCH":GOTO 10
```

Ist beim Basic 7.0 nach einer IF...THEN-Abfrage kein ELSE vorhanden, dann springt das Programm in die nächste Zeile. Bei der ELSE-Anweisung muß man darauf achten, daß vor dem ELSE ein Doppelpunkt steht. Befindet sich vor dem ELSE ein Leerzeichen, dann wird die ELSE-Anweisung nicht erkannt und auch nicht ausgeführt. Das Programm fährt einfach im Ablauf in der nächsten Zeile fort.

Wie springt man in Basic?

Die ELSE-Anweisung muß sich in der gleichen Programmzeile befinden, in der THEN steht. Da bleibt natürlich nicht viel Platz für Programmbefehle. Also doch wieder GOTO und GOSUB? Glücklicherweise nicht, den die IF...THEN-Anweisung ist noch weiter verbessert worden.

Um umfangreichere Schleifenbedingungen abzufangen, hat das Basic 7.0 des Commodore 128 die Kommandos BEGIN und BEND bekommen. Diese beiden Befehle dürfen aber nur im Zusammenhang mit der IF...THEN-Anweisung benutzt werden und müssen sich hinter dem THEN befinden. Damit kann man einem beliebig langen,



mehrere Programmzeilen umfassenden Befehlscode mit der Abfrage verknüpfen.

```
10 PRINT"HAST DU HUNGER? (J/N)"
20 GETKEY A$
30 IF A$="J" THEN BEGIN
40 PRINT"WARUM MACHST DU DIR
DANN NICHTS ZU ESSEN? DU"
50 PRINT"HAST WOHL VERGESSEN
EINZUKAUFEN?"
60 PRINT"DA KANN ICH DIR LEIDER
NICHT HELFEN."
70 BEND:ELSE:PRINT"ICH AUCH
NICHT"
80 END
```

Dieses kleine Programm fragt, ob Sie Hunger haben. Wenn ja, dann werden die Zeilen 40 bis 60 ausgeführt. Wenn Sie N oder ein anderes Zeichen als J eingeben, dann verzweigt das Programm nach Zeile 70. Dort wird mit BEND der Programmteil nach der THEN-Anweisung beendet.

Starke Hilfe durch BEGIN und BEND

Wie man sieht, darf auch bei IF...THEN mit anschließendem BEGIN...BEND noch ein nachfolgendes ELSE stehen. Hier finden wir also eine Ausnahme zu der oben beschriebenen Regel, daß sich das ELSE in der gleichen Programmzeile wie THEN befinden muß. Das ELSE muß sich aber direkt nach der BEND-Anweisung befinden und auch hier wieder den obligatorischen Doppelpunkt setzen. Nach dem ELSE kann nochmals ein BEGIN...BEND folgen.

Durch diesen erweiterten IF-Befehl kann in der Regel auf GOTO und GOSUB verzichtet werden. Völlig neu ist ein weiterer, sehr mächtiger Basic-Befehl: DO...LOOP.

Mit DO...LOOP lassen sich im Basic 7.0 Programmschleifen ohne GOTO erzeugen. Diese Anweisung ist mit FOR...NEXT zu vergleichen. Bei der FOR...NEXT-Anweisung muß jedoch von Anfang an die Anzahl der Schleifen-Durchläufe festgelegt sein.

```
10 FOR A=0 TO 10
20 PRINT"2 HOCH "A" ="2↑A
30 NEXT
```

Unser Beispiellisting gibt die ersten 11 Zweierpotenzen auf dem Bildschirm aus. Jetzt gibt es jedoch Fälle, in denen von vornherein nicht bekannt ist, wie oft eine Schleife auszuführen ist. Beispielsweise wenn man auf die Erfüllung einer Bedingung wartet, die durch den Schleifenablauf beeinflußt wird. In einem solchen Fall nimmt man DO...LOOP.

Programmteile zwischen DO und LOOP werden endlos wiederholt. Um jetzt keine »toten« Endlosschleifen zu erzeugen, braucht man eine Anweisung, um aus der Schleife wieder heraus zu kommen. Eine Form, die DO...LOOP-Schleife abzubrechen besteht im Basic-Befehl EXIT. Dieser Befehl wird häufig mit einer IF-Abfrage verknüpft sein. Findet das Programm in einer Schleife ein EXIT, so springt es direkt hinter den LOOP-Befehl und setzt dort das Programm fort.

Exit – der Basic-Aussteiger aus Endlos-Schleifen

```
10 INPUT"WIE GROSS DARF DIE
ZWEIERPOTENZ MAXIMAL SEIN";ZAHL
20 DO
30 C=A:A=2↑B:B=B+1
40 IF A>ZAHL THEN EXIT
50 LOOP
60 PRINT"DIE UNTER"ZAHL"LIEGENDE
ZWEIERPOTENZ"
70 PRINT"HAT DEN WERT VON
2 HOCH"B-1"="C
80 PRINT"DIE UEBER"ZAHL"LIEGENDE
ZWEIERPOTENZ"
70 PRINT"HAT DEN WERT VON
2 HOCH"B"="A
```

Mit diesem Programm wird eine Zweierpotenz gesucht, die nicht größer als die von uns eingegebene Zahl ist. Dazu wird die Schleife DO...LOOP so lange durchlaufen, bis die Abfrage in Zeile 40 feststellt, daß die Zweierpotenz den vorgegebenen Wert überschritten hat. Dann wird hinter die Schleife verzweigt und in den Zeilen 60 bis 80 werden die Ergebnisse ausgegeben.

Eine andere Form, die DO...LOOP-Schleife abzubrechen, findet man in der Befehlserweiterung UNTIL oder WHILE. Beide Befehle können nur als Zusatz zu DO....LOOP benutzt werden.

Bei UNTIL wird geprüft, ob eine vorgegebene Bedingung zutrifft. Die Schleife wird also so lange durchlaufen, bis (englisch: until) der nach UNTIL stehende Ausdruck logisch wahr wird, das heißt bis dieser Ausdruck stimmt.

```
10 PRINT"SPIELEN WIR ZAHLENRATEN
? (J/N)"
20 GETKEY A$
30 IF A$="J"THEN BEGIN
40 X=RND(1):X=INT(100*X)
50 BEND:ELSE PRINT"SCHADE,
TSCHUESS!":END
60 DO UNTIL A=X
70 INPUT"WIE HEISST MEINE ZAHL";A
80 IF A>X THEN PRINT"ZU GROSS"
90 IF A<X THEN PRINT"ZU KLEIN"
100 LOOP
110 PRINT"HURRA, RICHTIG."
120 END
```

Bei diesem kleinen Spiel muß eine zufällige Zahl zwischen 0 und 100 geraten werden. Gibt man auf die Frage des Computers ein J ein, dann wird diese zufällige Zahl in der Zeile 40 mit der RND-Funktion vom Computer errechnet. Wenn man nicht spielen will, verzweigt das Programm hinter das BEND in Zeile 50. Die Schleife beginnt mit dem DO in Zeile 60. In dieser Zeile wird abgefragt, ob schon die richtige Antwort eingegeben wurde. Ansonsten gibt der Computer in den Zeilen 80 und 90 Tips, ob die Zahl größer oder kleiner als der eingegebene Wert ist. Hat man das Ergebnis erraten, wird nach der Zeile 110 hinter der Schleife verzweigt.

Schleifen nach Maß – DO...LOOP

Dieses Spiel muß nach dem Erraten einer Zahl wieder mit RUN gestartet werden. Das kann man sich natürlich einfacher machen, indem man noch ein DO...LOOP um das ganze Programm setzt. Fügen Sie folgende Zeilen hinzu:

```
5 DO
120 LOOP
```

Jetzt ist das Programm in einer Endlosschleife gefangen, die nur durch das END in Zeile 50 sowie durch die RUN/STOP-Taste abgebrochen wird.

In unserem Programm sitzt die UNTIL-Abfrage direkt hinter dem DO. Sie kann aber auch hinter dem LOOP sitzen. In unserem Fall wäre das sogar besser, weil noch ein kleiner Fehler im Programm steckt. Wenn der Computer in der Zeile 40 zufällig die Zahl 0 nimmt, dann ist die Abfrage in Zeile 60 wahr, obwohl wir noch keine Zahl eingegeben haben. Der Unterschied in der Plazierung hat also eine große Wirkung. Im Fall des DO UNTIL wird die Schleife erst nach der Abfrage ausgeführt, bei DO...LOOP UNTIL erfolgt die Abfrage der Bedingung erst nach dem ersten Durchlauf der Schleife.

Deshalb muß unser Listing so aussehen:

```
5 DO
10 PRINT"SPIELEN WIR ZAHLENRATEN?
(J/N)"
20 GETKEY A$
30 IF A$="J"THEN BEGIN
40 X=RND(1):X=INT(100*X)
50 BEND:ELSE PRINT"SCHADE,
TSCHUESS!":END
60 DO
70 INPUT"WIE HEISST MEINE ZAHL";A
80 IF A>X THEN PRINT"ZU GROSS"
90 IF A<X THEN PRINT"ZU KLEIN"
100 LOOP UNTIL A=X
110 PRINT"HURRA, RICHTIG."
120 LOOP
```

Bei WHILE wird die Schleife so lange ausgeführt, wie die Bedingung hinter WHILE logisch wahr ist, das heißt solange es stimmt, was hinter WHILE steht. Im Gegensatz zu UNTIL wird bei WHILE also darauf gewartet, das eine Bedingung nicht mehr logisch wahr ist. Auch WHILE kann sowohl hinter DO als auch hinter LOOP stehen.

```
10 DO WHILE A<100
20 B=A*A:PRINT A
30 LOOP
```

Diese Form der Basic-Schleifengestaltung macht sowohl die Programmierung einfacher als auch ein Programm übersichtlicher. Für die Übersichtlichkeit gibt es aber auch noch einen anderen Weg.

Programmzusammenhänge sichtbar machen

Unter strukturiertem Programmieren kann man auch die Form des Listing-Aufbaues sehen. Es gilt allgemein als guter Programmierstil, wenn bestimmte Programmteile durch Einrücken des Textes auf dem Bildschirm sichtbar gemacht werden. Dadurch sieht man sofort, wie das Programm gegliedert ist, wo ein Programmteil anfängt und wo er endet. In Basic ist diese sichtbare Strukturierung nur bei der Arbeit am 80-Zeichen-Bildschirm sinnvoll, da bei 40-Zeichen-Darstellung jeweils zwei Bildschirmzeilen für eine Programmzeile zur Verfügung stehen. Da nützt auch die schönste Einrückung von Programmzeilen nichts. Dasselbe Programm auf dem 80-Zeichen-Monitor wirkt dagegen aufgeräumt und übersichtlich. Bei dem Einrücken der Zeilen hilft die TAB-Taste. Bei einem Druck auf die TAB-Taste springt der Cursor jeweils in 8-Zeichen-Schritten weiter. Man muß nicht immer wieder eine Reihe von Leerzeichen eingeben. Allerdings ist zu Beginn jeder Zeile ein Doppelpunkt notwendig, weil der Basic-Editor des C 128 überflüssige Leerzeichen nach der Zeilennummer ignoriert. Er setzt die Befehle an den Zeilenanfang zurück. Mit dem Doppelpunkt wird jedoch der Zeilenanfang definiert und nachfolgende Leerzeichen als Anweisung ohne Wirkung interpretiert.

Lust auf Basic

Sicherlich ist es nicht leicht, in Basic übersichtlich zu programmieren. Sowohl die sichtbare als auch die logische Struktur erfordert Programmiererfahrung. Wer sich daran gewöhnt hat, wird die Vorteile schnell erkennen. Man spart Zeit und Nerven.

Das Basic 7.0 des Commodore 128 ist ein sehr wirksames Werkzeug, sowohl in der Ausnutzung der Fähigkeiten dieses tollen Computers als auch in der Art, wie sich das Basic programmieren läßt. Programme schreiben macht auf jeden Fall auf dem Commodore 128 mit seinem Basic 7.0 sehr viel Spaß.

(zu)

Quelle: Basic 7.0 auf dem Commodore 128, Markt&Technik Verlag AG, ISBN 3-89090-170-0

```
10 GRAPHIC 1,0:SCNCLR
20 COLOR 0,1: COLOR 1,2
30 FOR K=0 TO 2*π + 0.05
   STEP 0.01
40 X=150+10-150*COS(3*K)
45 Y=100-150*0.5*SIN(5*K)
50 DRAW,X,Y TO X,Y
60 NEXT
```

Listing 1. Grafik auf dem C128

```
10 POKE 53272,PEEK(53272)OR 8:POKE
   646,0
20 POKE 53265,PEEK(53265)OR 32
30 FOR K=0 TO 999:POKE 1024+K,14:N
   EXT
40 FOR I=0 TO 7999:POKE 8192+I,0:N
   EXT
50 FOR K=0 TO 2*π+0.05 STEP 0.01
60 X=150+10-150*COS(3*K)
70 Y=100-150*0.5*SIN(5*K)
80 FOR N=0 TO 24
90 IF Y>N*8-1 AND Y<(N+1)*8 THEN B
   Y=8192+N*320+8*INT(X/8)+Y-8*N:N
   =24:GOTO 110
100 NEXT N
110 BI=8*(1+INT(X/8))-X-1
120 BY=INT(BY+0.5)
130 POKE BY,PEEK(BY)OR 2↑INT(BI+0.
    5)
140 NEXT:NEX:
150 GET A$:IF A$="" THEN 140
```

Listing 2. Dieselbe Grafik auf dem C64

```
10 VOL 15
20 ENVELOPE 0,8,8,6,0,1
30 TEMPO 30
40 PLAY "V1 T0 04 QCQDQEQFHGHG QAQA
   AQAQAHGR QAQAQAQAHGR QFQFQFQFHEHE
   QDQDQDQDHC"
```

Listing 4. Dasselbe Lied im 7.0 Basic

```
10 A$="HAMBURGER":B$="SENF":C$="KE
   TCHUP":D$="SALAT":E$="FLEISCH":
   F$="BROETCHEN"
11 INPUT"WAS MOECHTEN SIE BITTE";Z
   $
12 IF Z$=A$THEN GOTO 16
13 IF Z$<>A$THEN PRINT"HABEN WIR L
   EIDER NICHT. WIE WAERS MIT EINE
   M HAMBURGER?"
14 GOTO 11
15 PRINT"KOMMEN SIE MORGEN WIEDER,
    MIR WIRD UEBEL":GOTO 24
16 INPUT"NA PRIMA, WAS GEHOERT DAZ
   U";Z$
17 IF Z$=B$THEN PRINT"WIE DAS SCHM
   ECKT!":GOTO 23
18 IF Z$=C$THEN PRINT"DARF NICHT F
   EHLEN!":GOTO 23
19 IF Z$=D$THEN PRINT"SIE SIND EIN
    KENNER!":GOTO 23
20 IF Z$=E$THEN GOTO 15
21 IF Z$=F$THEN PRINT"DAMIT ER NIC
   HT SO IN DER HAND RUMFLUTSCHT!"
   :GOTO 23
22 PRINT"ABER WIRKLICH NICHT!"
23 INPUT"WAS NOCH";Z$:GOTO 17
24 FOR A=1 TO 3000:NEXT:GOTO 10
```

Listing 5. »Hamburger«

```
10 SI=54272
20 FOR L=SI TO SI+24:POKE L,0:NEXT
30 POKE SI+5,9:POKE SI+6,0
40 POKE SI+24,10
50 READ HB,LB,DR
60 IF HB<0 THEN POKE SI+24,0
70 POKE SI+1,HB:POKE SI,LB
80 POKE SI+4,33
90 FOR Z=1 TO DR:NEXT
100 POKE SI+4,32:FOR Z=1 TO 50:NEX
    T
110 GOTO 50
200 DATA 17,103,250
210 DATA 19,137,250
220 DATA 21,237,250
230 DATA 23,059,250
240 DATA 26,020,500
250 DATA 26,020,500
260 DATA 29,069,250
270 DATA 29,069,250
280 DATA 29,069,250
290 DATA 29,069,250
300 DATA 26,020,500
310 DATA 29,069,250
320 DATA 29,069,250
330 DATA 29,069,250
340 DATA 29,069,250
350 DATA 26,020,500
360 DATA 23,059,250
370 DATA 23,059,250
380 DATA 23,059,250
390 DATA 23,059,250
400 DATA 21,237,500
410 DATA 21,237,500
420 DATA 19,137,250
430 DATA 19,137,250
440 DATA 19,137,250
450 DATA 19,137,250
460 DATA 17,103,500
470 DATA -1, -1. -1
```

Listing 3. »Alle meine Entchen« auf dem C64


64er

Software für den C 128

Während die einen vom Schlagwort CP/M fasziniert sind, warten die anderen auf Software für den C128-Modus. Wir stellen vor, was es im Augenblick gibt und was es in den nächsten Monaten geben wird.

Wenn beim C 128 die Rede auf professionelle Software kommt, dann hört man meistens nur drei Buchstaben: CP/M. Dabei ist die CP/M-Software nicht das Optimum an Leistungsfähigkeit. Denn CP/M-Software muß ja auf sehr vielen Computern laufen und kann somit nicht die einzelnen Computer voll ausnutzen. Anders ist dies bei Software-Produkten für den C 128-Modus. Diese sind in der Regel schneller, leistungsfähiger und einfacher zu bedienen als CP/M-Programme. Dafür kauft man sich den Nachteil ein, völlig an den C 128 gebunden zu sein, da die Programme und die damit verarbeiteten Daten nicht ohne Probleme an andere Computer zu übertragen sind.

Im folgenden werden wir deswegen in einem kleinen Rundumschlag die Software vorstellen, die für den C 128 erhältlich ist oder in Kürze erhältlich sein wird.

Einfach super: Superbase und Superscript

Zwei Softwareprodukte, die im Gespann das Präfix »Super« verdienen, sind »Superbase«, ein Datenverwaltungsprogramm, und »Superscript«, ein Textverarbeitungsprogramm. Beides sind unabhängige Einzelprodukte, die aber auch gemeinsam genutzt werden können.

»Superbase« ist vielen Besitzern der älteren Commodore-Computer-Besitzer sicherlich bekannt. Für den C 64 ist »Superbase« das beste erhältliche Programm seiner Art. »Superbase« ist ein komplettes Datenverarbeitungssystem: Wer nur seine Daten eingeben und ausgeben und etwas bearbeiten will, wird genauso versorgt wie derjenige, der die verrücktesten Sachen machen will. Denn »Superbase« hat eine Programmier-Option, mit der man Report-Generatoren, verbesserte Eingabemöglichkeiten, automatische Datenpflege, Listen-Verarbeitung und vieles weitere mehr machen kann. Und selbst wer überhaupt keine Ahnung von Computern im allgemeinen und Datenbanken im besonderen hat, wird mit »Superbase« zurechtkommen. Ein sehr ausführliches deutschsprachiges Handbuch vermittelt in einem Übungsteil mit sehr vielen Beispielen, vom Einschalten des Computers und Laden des Programms bis hin zur Report-Erstellung über einzelne Datensätze, alle Fähigkeiten von »Superbase«. Auf der Programm-Diskette sind alle Übungsbeispiele enthalten.

Die Textverarbeitung »Superscript« glänzt mit ähnlichen Merkmalen wie »Superbase«. Von der einfachen Büro-Notiz bis zum Serien-Geschäftsbrief – »Superscript« bietet alle Funktionen, die man sich wünscht. Alle Funktionen können über Menüs oder Kommandos angewählt werden. Außerdem bietet »Superscript« völlig frei belegbare Befehlstasten an, die man mit beliebigen Befehlskombinationen, Textbausteinen oder Formatier-Anweisungen für einen Drucker belegen kann. »Superscript« kann auch kleinere Kalkulationen im Text ausführen. Damit wird die Erstellung von Tabellen und Rechnungen zum Kinderspiel. Auch für »Superscript« gibt es so ausführliche und verständliche Handbücher wie für »Superbase«.

»Superbase« und »Superscript« können gleichzeitig im Speicher des C128 stehen. Dann kann zwischen beiden Programmen hin- und hergeschaltet werden. Dies läßt sich für Serienbriefe, komfortablere Datenpflege und vieles andere mehr sehr sinnvoll einsetzen. Im Team sind die beiden augenblicklich unschlagbar und stellen wohl das beste Programmpaket für den C 128 dar. Beide Programme sind jetzt schon lieferbar und kosten jeweils 198 Mark. In Deutschland wird der Vertrieb von Commodore übernommen.

Natürlich sind dies nicht die beiden einzigen Programme, die es für den C 128 geben wird. Viele andere Softwarefirmen werden Software für den C 128 herausbringen, die allerdings augenblicklich noch nicht fertiggestellt ist.

Weitere Profi-Programme

Eines dieser Produkte ist »Vizawrite«. Im Augenblick wird von diesem Programm, das als beste Textverarbeitung für den C 64 bezeichnet wird, eine neue Version für den C 128 erstellt. Dabei sollen die guten Leistungen von »Vizawrite 64« noch einmal gesteigert werden. Voraussichtlicher Preis: 348 Mark.

Natürlich hat auch der rot-weiße Riese aus Düsseldorf schon ein paar Eisen im Feuer: Data Becker wird demnächst C 128-Versionen der Programme »Datamat«, »Textomat« und »Textomat plus« anbieten. Alle Programme sind schon bei vielen C 64-Besitzern in Betrieb und haben sich dort bewährt. Die C 128-Versionen sollen nicht nur an den C 128 angepaßt, sondern auch noch verbessert werden. Der »Textomat 128« wird übrigens mehr können als der »Textomat Plus« für den C 128.

Eine weitere Umsetzung eines vom C 64 und von den »großen« CBMs her bekannten Programms ist »Protext 128«. »Protext« war bisher auf dem C 64 nur mit einer 80-Zeichen-Karte lauffähig, doch die ist ja im C 128 als neuer Videochip gleich enthalten. »Protext« ist »Superscript« in vielen Punkten überlegen (Geschwindigkeit, Funktionen) und kostet dennoch nur 149 Mark. Eine dazu passende Dateiverwaltung ist in Vorbereitung.

SM-Software bereitet die gesamte SM-Manger-Serie (Lager, Rechnung, Kunden) und den bekannten SM-Text für den C 128 auf.

Programmiersprachen

Kaum ist der C 128 mit seinem guten Basic 7.0 auf dem Markt, gibt es schon zwei Basic-Compiler zu kaufen. Der eine ist »Austrocomp/Austrospeed 128« von der Firma Digimat. Er wird wahrscheinlich über Commodore vertrieben. Der andere Compiler ist »Basic 128«, eine neugeschriebene Version des »Basic 64« von Data Becker. Beide Compiler unterstützen das neue Basic 7.0 und die meisten neuen Möglichkeiten des C 128.



Aber auch andere Programmiersprachen sind im anrollen: Data Becker hat neue Versionen von »Profi-Pascal« und »C-Compiler« angekündigt, die ebenfalls voll auf den Commodore 128 zurechtgeschnitten sind.

Unterhaltungssoftware

Ein Bereich, auf dem sich im Gegensatz zum C 64 recht wenig tut, ist der der Unterhaltungssoftware, sprich Spielen, Musikprogrammen und ähnlichem. Ein einziges Spiel wurde für den C 128 angekündigt: »Gato« ist eine Zweiter-Weltkrieg-U-Boot-Simulation. Allerdings ist hier noch nicht ganz geklärt, wann und ob überhaupt eine C 128-Version herauskommen wird. Ansonsten muß man bei Spielen auf den bewährten C 64-Modus zurückgreifen, der von »Summer Games« bis »Skyfox« alles bietet, was das Herz begehrt.

Schon lieferbar ist der »Music Maker 128« von Music Sales Ltd., der, wie der Name schon sagt, ein Musik-Programm ist. Hier wurde gegenüber der C 64-Version einiges an Verbesserungen vorgenommen.

Zukunftsaussichten

Ein besonders interessantes Produkt wird von Softline angeboten: »Newsroom« ist ein Programm, um Zeitungen mit dem C 64 oder C 128 herzustellen. Das besondere: »Newsroom« läuft auf beiden Computern und paßt sich dem verwendeten Computer an: Auf dem C 128 steht mehr Speicherplatz zur Verfügung, der dann auch vom Programm ausgenutzt wird.

Was bringt uns die Zukunft an Software für den C 128? Nun, sehr viel kann man hier nicht sagen. Denn die meisten Software-Hersteller sind sehr skeptisch, was den C 128 betrifft. Und die Produkte, die im Augenblick lieferbar sind, sind mehr oder minder gelungene Umsetzungen von C 64-Software. Ein Markt, auf dem es sehr öde aussieht, ist der Unterhaltungssektor. Denn schließlich lohnt es sich bei der riesigen Anzahl von verkauften C 64 kaum, ein Programm nur für den C 128-Modus herauszubringen. Da wird man wohl weiterhin auf den C 64-Modus angewiesen sein. (Warum auch nicht!).Dafür sind professionelle Programme und Programmiersprachen hoch im Kurs. Aber auch hier stellen sich manche Hersteller die Frage, ob der Markt vorerst nicht mit den schon erschienenen, sehr guten Produkten abgedeckt ist. Sensationen sind vorerst nicht zu erwarten. (bs)

Info:
Protext: Markt&Technik Verlag, Hans-Pinsel-Str. 2, 8013 Haar
Superscript & Superbase:
Commodore Büromaschinen GmbH, Lyonerstr. 38, 6000 Frankfurt 71
Textomat, Datamat, Profi-Pascal, C-Compiler und Basic 128:
Data Becker, Merowingerstr. 30, 4000 Düsseldorf
SM-Software, Fasangartenstr. 4, 8000 München 83
Vizawrite:
Interface Age, Josephsburgstr. 6, 8000 München 80
Austrocomp & Austrospeed
Digimat, Arbeitergasse 48, A-1050 Wien
Newsroom, Gato:
Softline, Schwarzwaldstr. 8a, 7602 Oberkirch
Music Maker:
SFX-Software, Wilhelmstr. 26, 5000 Köln 90

Der C 128 am Telefon

Über Akustikkoppler oder Modem mit Mailboxen in Verbindung treten. Das kann der Commodore 128 auch!

Im C 64-Modus behandelt der Commodore 128 den User-Port und damit die RS232-Schnittstelle wie gewohnt. Gute Bekannte wie das Datenfernübertragungs-Programm »Proterm 64« (64'er Sonderheft 7/1985) versehen einwandfrei ihren Dienst. Wie aber verhält sich der User-Port im 128-Modus?

Der User-Port des C 128 wird vorwiegend für den Druckeranschluß oder für den Anschluß eines Akustikkopplers genutzt. Dazu wird entweder eine parallele oder serielle Schnittstelle am User-Port simuliert. Um Daten mit dem Telefon zu übertragen, ist meist die serielle RS 232-Schnittstelle erforderlich. Mit dieser Schnittstelle werden acht Byte nicht gleichzeitig nebeneinander, sondern Bit für Bit hintereinander gesendet. Dadurch wird zwar die Geschwindigkeit wesentlich geringer als bei der parallelen Datenübertragung, aber man kommt mit viel weniger Leitungen aus. Und gerade im Telefon-Verkehr ist die serielle Übertragung die einzig mögliche.

Der Commodore 128 ist von Haus aus im Betriebssystem mit der erforderlichen Software zum Betrieb einer seriellen RS232-Schnittstelle ausgerüstet. Die RS232-Schnittstelle selbst muß noch mit einem Modul hardwaremäßig am User-Port simuliert werden. Dazu gibt es in der 64'er, Ausgabe 3/85 einen kleinen Bausatz für 25 Mark, der den Normpegel am User-Port (TTL-Pegel von +5V) auf den RS232-Pegel (±3V bis ±15V) umwandelt.

Die RS232-Schnittstelle hat im C 128 die Geräteadresse 2 bekommen. Dadurch wird diese Schnittstelle wie ein Floppy-Laufwerk oder ein Drucker angesprochen.

Eröffnet man beim Commodore 128 ein User-Port-File mit der OPEN-Anweisung, so legt der C 128 zwei Daten-Buffer von je 256 Byte an. In den Eingabebuffer, der ab der Adresse $0C00 Hex liegt, werden alle über die RS232-Schnittstelle hereinkommenden Daten gesammelt. Der Ausgabebuffer ab der Adresse $0D00 nimmt die an die RS232-Schnittstelle auszugebenden Daten auf.

Um die Datenübertragung zu organisieren gibt es zwei spezielle Register: das Kontrollregister und das Befehlsregister. Diese werden automatisch mit der Eröffnung des logischen Files (Geräteadresse 2) definiert.

Das Befehlsregister bestimmt die Übertragungsart, die Parität und die Art der Verbindung (Handshake). Im Kontrollregister wird die Baud-Rate definiert und die Anzahl der Daten- und Stop-Bits.

Die Datenübertragungsrate beträgt bei den meisten C 64-Mailboxen 300 Baud bei sieben Daten-Bits und einem Stop-Bit. Das Stop-Bit kennzeichnet das Übertragungsende von Daten-Bits. Die Baud-Rate darf beim C 128 den Wert von 2400 nicht übersteigen, da die interne Software des C 128-Betriebssytems nicht schneller arbeitet.

Alle Parameter werden in der OPEN-Anweisung übergeben. Die Syntax für den OPEN-Befehl lautet: »OPEN Filenummer, Geräteadresse, Sekundäradresse, Kontrollregister, Befehlsregister«. Eine Eröff-

nung des Datenkanals mit allen Parametern sieht beispielsweise folgendermaßen aus:

»OPEN 2,2,0,CHR$ (6+32+0)+CHR$(0+0+0)«

Damit haben wir folgende Parameter programmiert:
- logisches File 2 öffnen
- Geräteadresse 2 ansprechen
- Sekundäradresse 0 setzen
- 300 Baud Übertragung
- 7 Daten-Bits
- 1 Stop-Bit
- 3-Draht-Leitung
- Vollduplex
- keine Parität (kein 8. Daten-Bit)

Bits 5,6 und 7 die Parität bestimmt (siehe Tabelle 2).

Selbstverständlich können bei der Datenübertragung mit der RS 232-Schnittstelle Fehler vorkommen. Dafür gibt es das sogenannte Status-Register, das allerdings nach dem Auslesen sofort gelöscht wird. Man sollte diese Statusvariable ST immer in einer Variablen ablegen. Die Bedeutung der einzelnen Bits der Statusvariablen ST sind der Tabelle 3 zu entnehmen.

Das Kontrollregister, das Befehlsregister und die Statusvariable kann man auch direkt aus der Zeropage auslesen. Für die RS232-Schnittstelle sind die Adressen $0A0F Hex bis $0A1B Hex wichtig (siehe Tabelle 4).

Daß ein Programm für die RS232-Schnittstelle nicht umfangreich sein muß, beweist unser kleines C 128-Terminalprogramm (siehe Listing). Mit ihm kann mit jeder Mailbox kommuniziert werden.

Leider ist der Basic 7.0 Interpreter etwas langsamer als der Dateneingang über die RS232-Schnittstelle. Wenn kontinuierlich Daten hereinkommen, läuft der Datenpuffer nach einiger Zeit über, weil der Computer mit dem Auslesen nicht mehr nachkommt. Die Folge ist, daß der Text plötzlich unleserlich wird. Erst wenn der Partnercomputer eine Pause macht, kann die Basic-Routine nach einem Moment wieder »vernünftigen« Text liefern.

Wesentlich mehr Komfort und Geschwindigkeit bringen in Maschinensprache geschriebene Terminalprogramme. Da nur kleine Änderungen der Speicheradressierung vorzunehmen sind, wird sicherlich demnächst mit mehreren Datenfernübertragungsprogrammen zu rechnen sein, die auch im C 128-Modus des Commodore 128 laufen. Nur steht diesen dann wesentlich mehr Speicherplatz für die Protokollführung zur Verfügung. Dann kann man sich mit dem C 128 Programme übertragen lassen, für die der Speicher des C 64 zu klein war. (zu)

Bit 3 2 1 0	dezimal	Baud-Rate
0 0 0 0	0	Anwender-Baud-Rate,
0 0 0 1	1	50
0 0 1 0	2	75
0 0 1 1	3	110
0 1 0 0	4	134.5
0 1 0 1	5	150
0 1 1 0	6	300
0 1 1 1	7	600
1 0 0 0	8	1200
1 0 0 1	9	1800
1 0 1 0	10	2400
1 0 1 1	11	3600
1 1 0 0	12	4800
1 1 0 1	13	7200
1 1 1 0	14	9600
1 1 1 1	15	19200
Bit 6 5	dezimal	Anzahl der Datenbits
0 0	0	8 Bit
0 1	32	7 Bit
1 0	64	6 Bit
1 1	96	5 Bit
Bit 7	dezimal	Anzahl der Stoppbits
0	0	1 Stoppbit
1	128	2 Stoppbit

Tabelle 1. Die Bit-Funktionen des Kontrollregisters

Bit 0	dezimal	Handshake
0	0	3-Draht-Handshake
1	1	X-Draht-Handshake
Bit 4	dezimal	Übertragungsart
0	0	Vollduplex
1	16	Halbduplex
Bit 7 6 5	0	Paritätsprüfung
X X 0	0	keine Paritätsprüfung kein 8. Datenbit
0 0 1	32	ungerade Parität
0 1 1	96	gerade Parität
1 0 1	160	8. Datenbit immer 1 keine Paritätsprüfung
1 1 1	224	8. Datenbit immer 0

Tabelle 2. Bit-Belegung des Befehlsregisters

Bit	Beschreibung
0	Paritätsfehler
1	Rahmenfehler
2	Empfängerbuffer voll
3	Empfängerbuffer leer
4	CTS (Clear To Send) Signal fehlt
5	unbenutzt
6	DSR (Data Set Ready) Signal fehlt
7	Break-Signal empfangen

Tabelle 3. Die Statusvariablen ST

Hex	Dezimal	Bedeutung
0A0F:	2575	RS-232 NMI Status Register
0A10:	2576	RS-232 Kontrollregister
0A11:	2577	RS-232 Kommandoregister
0A12:	2578 - 2579	RS-232 Benutzer Baud-Rate
0A14:	2580	RS-232 Statusregister
0A15:	2581	RS-232 Anzahl der zu sendenden Bits
0A16:	2582 - 2583	RS-232 Baud-Rate: Full Bit Time (in µs)
0A18:	2584	RS-232 Index auf den Anfang des Eingabepuffers
0A19:	2585	RS-232 Index auf das Ende des Eingabepuffers
0A1A:	2586	RS-232 Index auf den Anfang des Ausgabepuffers
0A1B:	2587	RS-232 Index auf das Ende des Ausgabepuffers

Tabelle 4. RS232-Adressen in der Zeropage

Wie man sieht, wird das Befehls- und Kontrollregister mit dem OPEN-Befehl definiert. Dazu haben wir nach der Adreß-Definition zwei Byte mit Hilfe der CHR$-Anweisung übergeben.

Einige Bits dieser beiden Register haben wichtige Funktionen. Die ersten vier Bit des Kontrollregisters bestimmen die Baud-Rate, Bit 5 und 6 die Anzahl der Datenbits und Bit 7 die Anzahl der Stop-Bits (siehe Tabelle 1).

Das zweite Byte wird in das Befehlsregister geschrieben und bestimmt mit Bit 0 die Art des Handshake-Betriebs. Die darauffolgenden drei Bit sind von untergeordneter Bedeutung. Mit Bit 4 wird die Übertragungsart und über die Bits 5,6 und 7 die Parität bestimmt (siehe Tabelle 2).

Quelle: Commodore 128 Intern von Data Becker, ISBN 3-89011-098-3

Das Commodore 128 Handbuch von Markt&Technik Verlag, ISBN 3-89090-171-9

```
1 OPEN 2,2,0,CHR$(6+32+0)+C
  HR$(0+0+0):GET#2,A$         <020>
2 PRINT CHR$(14);"{CLR,DOWN
  ,2SPACE}DATENFERNUEBERTRA
  GUNG MIT DEM C 128{DOWN}"  <027>
3 GET A$:IF A$=""THEN 7       <032>
4 C%=0:D%=ASC(A$):IF D%<91
  AND D%>64 THEN C%=32        <038>
5 IF D%=20 THEN D%=8          <124>
6 D%=D%+C%:PRINT#2,CHR$(D%)
  ;                           <130>
7 GET#2,B$:IF B$=""THEN 3     <104>
8 C%=0:D%=ASC(B$):IF D%<91
  AND D%>64 THEN C%=128       <024>
9 IF D%>96 THEN D%=D%-32      <063>
10 IF D%=8 THEN D%=20         <207>
11 D%=D%+C%:PRINT CHR$(D%);
   :GOTO 3                    <001>
```

Listing. Ein Terminalprogramm für den C 128



Der Basic-Interpreter des C 128

Um Ihnen Arbeit und schlaflose Nächte zu ersparen, zeigen wir, wie Sie beim C 128 die Basic-Interpreter-Routinen in eigene Maschinenprogramme einbauen können.

Zur Programmierung von Assemblerunterroutinen für Basic-Programme benötigt man oft die Basic-Interpreter-Routinen zur Parameterübergabe, zum Anlegen von Variablen und zum Rechnen mit Integer- und Realzahlen. In diesem Artikel wird gezeigt, wo sich diese Routinen beim C 128 befinden und worauf bei ihrem Einsatz zu achten ist.

Reine Maschinenprogramme kommen üblicherweise ohne jede Stringbehandlung aus. Bei einer Textverarbeitung zum Beispiel liegt der Text am Stück im Speicher und der Programmierer verwaltet selbst seine Zeiger auf die momentane Textstelle, auf die Seitenanfänge und so weiter.

Problematischer sind jedoch Unterprogramme in Maschinensprache, die zeitkritische Basic-Teile ersetzen sollen. In diesen Fällen soll oftmals das Ergebnis der Maschinenroutine in Form einer Variablen an das aufrufende Basic-Programm zurückübergeben werden. Wenn die Routine komfortabel aufzurufen sein soll, kann das Basic-Programm beim Aufruf mit »SYS....,(Variablenname)« angeben, in welcher Variablen das Ergebnis abgelegt werden soll.

In beiden Fällen, der Übergabe einer Variablen an das Maschinenprogramm und der Rückübergabe des Ergebnisses in Form einer Variablen, benötigen wir verschiedene Routinen des Basic-Interpreters. Es gibt eine Vielzahl von Anwendungen, in denen diese Routinen benötigt werden:

a) Sortierroutinen, die in der Lage sein sollten, beliebige Arraytypen zu sortieren und denen der Name des gewünschten Arrays übergeben werden muß. Weiterhin müssen Sortierroutinen über noch zu besprechende Routinen des Basic-Interpreters Anfang und Ende dieses Arrays erkennen können.

b) die beliebten INPUT-Routinen, die den eingebauten Befehl »INPUT X$« ersetzen sollen. Komfortablen INPUT-Routinen sollen zum Beispiel folgende Parameter übergeben werden können: Spalte und Zeile, an der die Eingabe beginnt; Länge des Eingabefeldes; als Eingabe zugelassene Zeichen; Zeichen, die die Eingabe beenden; String, in dem die Eingabe an das Basic übergeben werden soll. Eine INPUT-Routine darf die Eingabe selbst natürlich nicht an beliebiger Stelle im Speicher ablegen, sondern muß einen String anlegen, der die Eingabe enthält und den das Basic-Programm weiterverarbeiten kann.

c) Suchroutinen, die ein beliebiges Array (Strings oder numerische Variablen) nach einer vorgegebenen Zeichenkette oder Zahl durchsuchen sollen, zum Beispiel die im Speicher gehaltene Indexdatei in einer Dateiverwaltung. In diesem Fall muß die Assemblerroutine auf das entsprechende Array zugreifen (womöglich sogar erst ab einem vorgegebenen Index, der den Start der Routine bestimmen soll), jedes Arrayfeld mit dem Suchkriterium vergleichen und nach Abschluß der Suche eine numerische Variable anlegen, der das Ergebnis mitgeteilt wird, zum Beispiel »x% = 5« (das fünfte Arrayelement entspricht dem übergebenen Suchkriterium).

Da ich selbst kürzlich vor der Aufgabe stand, solche und andere Routinen vom C 64 auf den C 128 umzuschreiben und daher gezwungen war, die besprochenen Basic-Interpreter-Routinen des C 128 zu untersuchen, bietet es sich an, mein frisch erworbenes Wissen weiterzugeben. Es dürfte manchem Assemblerprogrammierer schlaflose Nächte ersparen, in denen er sich anhand des eingebauten Monitors durch das Basic-ROM des C 128 »hindurchwühlt«.

Bevor wir Strings oder Variablen anlegen können, ist es nötig, alle Arten von Parametern, die das aufrufende Basic-Programm übergibt, aus dem Basic-Text herauszulesen. Der C 128 bietet hier einiges mehr an Komfort als der C 64, da der SYS-Befehl bereits für die Übergabe von Parametern vorgesehen ist. Er besitzt folgendes Format:

SYS (Startadresse),(Akku),(X-Register),(Y-Register),(Status)

Beim Aufruf können daher bereits Ein-Byte-Werte übergeben werden, mit denen der Akku und die verschiedenen Register geladen werden. Im weiteren Verlauf dieses Artikels wird das Statusregister jedoch nicht verwendet werden. Bedenken Sie, daß die Abfrage des Statusregisters nur unter »Verrenkungen« möglich ist, indem die verschiedenen Flaggen getestet werden, wobei bei einigen Bits ein Test nicht möglich ist. Ich werde mich daher auf die Übergabe von maximal drei Werten im Akku, im X- und im Y-Register beschränken.

Ein Beispiel: Geben Sie mit dem eingebauten Monitor ein:

```
A 0B00 BRK
```

Geben Sie nun im Direktmodus ein:

```
SYS DEC("0B00"),1,2,3
```

Nach dem BRK wird der Monitor die übergebenen Werte in den Registern anzeigen:

Akku: 1; X-Register: 2; Y-Register: 3

Sie sind keineswegs gezwungen, beim Aufruf immer alle Parameter anzugeben. Wollen Sie zum Beispiel nur im Akku den Wert 20 und im X-Register eine 53 übergeben, genügt der Aufruf mit:

```
SYS DEC("0B00"),20,53
```

Soll ein Register »übergangen«, also nicht mit einem definierten Wert geladen werden, können Sie den entsprechenden Parameter einfach weglassen.

Beispiel: Akku mit 15 und Y-Register mit 87 laden:

```
SYS DEC("0B00"),15,,87
```

Für viele Anwendungen ist diese Form der Übergabe jedoch unzureichend, zum Beispiel für die Übergabe eines Zwei-Byte-Wertes oder einer Stringvariablen. Auch dann, wenn zwar ausschließlich Ein-Byte-Werte übergeben werden sollen, die Anzahl der Parameter jedoch größer drei ist, benötigen wir Routinen, die uns weitere Parameter aus dem Basic-Text holen. Die erste und einfachste dieser Routinen ist GETBYT, die einen Ein-Byte-Wert aus dem Basic-Text holt und im X-Register übergibt.

Bevor ich nun die Routine GETBYT beschreibe, geben Sie bitte die folgende Programmzeile ein, speichern Sie das Programm, und starten Sie es:

```
10 POKE DEC("2E"),DEC("20"):POKE
DEC("2000"),0:POKE DEC("30"),DEC
("20"):CLR:NEW
```

Gehen Sie nun in den eingebauten Monitor und geben Sie folgende Zeilen ein:

Initialisierung

```
a 00b00  ad 00 ff lda $ff00
a 00b03  48       pha
a 00b04  a9 00    lda #$00
a 00b06  8d 00 ff sta $ff00
a 00b09  ad 06 d5 lda $d506
a 00b0c  09 06    ora #$06
a 00b0e  8d 06 d5 sta $d506
a 00b11  68       pla
a 00b12  8d 00 ff sta $ff00
a 00b15  60       rts
```

Speichern Sie auch dieses Programm und rufen Sie es mit »SYS DEC("0B00")« auf.

Ein Problem bei der Programmierung des C 128 in Assembler ist das Bankswitching. Die Routinen des Basic-Interpreters schalten ständig zwischen Bank 0 (Basic-Text) und Bank 1 (Variablenbank) um. Unsere Programme legen wir in Bank 0. Wenn wir nun eine Routine des Basic-Interpreters aufrufen, nach deren Ausführung auf Bank 1 umgeschaltet ist, ist unser in Bank 0 liegendes Programm für den Prozessor nicht existent. Die Folge ist zwangsläufig ein Absturz.

Das Initialisierungsprogramm hat die Aufgabe, den Bereich $0000 - $1FFF als gemeinsamen Speicherbereich zu definieren. Egal, ob bei der Rückkehr aus einer Interpreter-Routine Bank 0 oder Bank 1 eingeschaltet ist. Unser Programm existiert immer in der jeweiligen Bank und kann daher weiter ausgeführt werden.

Gewöhnen Sie sich auf alle Fälle an, diese Initialisierung (sowohl das Basic- als auch das Maschinenprogramm) nach jedem Einschalten des Rechners oder Reset als erstes zu laden und aufzurufen. Sie vermeiden dadurch Abstürze, die bei Betrachtung der aufgerufenen Assemblerroutine selbst völlig unerklärlich sind und nur durch das Bankswitching zustande kamen.

CHKKOM ($795C)

Bevor wir, wie beabsichtigt, einen zusätzlichen Parameter aus dem Basic-Text holen, benötigen wir die Routine CHKKOM. Sie liest das nächste Zeichen des Basic-Textes (ausgehend von dem momentanen Stand des Programmzeigers der CHRGET/CHRGOT-Routine) und inkrementiert den Programmzeiger. Bei dem gelesenen Zeichen wird überprüft, ob es sich um ein Komma handelt. Wenn ja, ist alles in Ordnung und es erfolgt ein RTS. Handelte es sich um ein anderes Zeichen, wird in die Routine zur Fehlerausgabe gesprungen und ein »SYNTAX ERROR« ausgegeben. Da es üblich ist, zur Trennung mehrerer Übergabeparameter – ebenso wie die SYS-Routine – Kommata zu benutzen, bietet es sich an, diese Routine vor dem Aufruf von GETBYT zu verwenden.

1. Ein Komma einlesen:

```
a 0b20  20 5c 79 jmp $795c
```

Nach dem Aufruf mit »SYS DEC("0B20"),0,0,0,0,« oder »SYS DEC("0B20"),,,,,« erhalten wir die Meldung READY. Der SYS-Befehl hat alle vier Parameter gelesen, anschließend erfolgte der Sprung zu unserem eigenen Assemblerprogramm, das ein zusätzliches Komma las und danach die Rückkehr ins Basic. Ein zusätzlicher Befehl RTS entfällt, da sich dieser am Ende des Unterprogramms CHKKOM befindet. Beachten Sie bitte, daß bei der Verwendung der Interpreter-Routinen zur Übergabe von Parametern zuerst alle vier möglichen Parameter, die der SYS-Befehl einlesen kann, angegeben werden müssen. Würden wir unser Programm zum Beispiel mit »SYS DEC("0B20"),« aufrufen, wäre ein SYNTAX ERROR die Folge, da der SYS-Befehl nach dem Komma den ersten Standardparameter (Akku) lesen will und nicht findet.

2. Drei Kommas einlesen:

```
a 00b20  20 5c 79 jsr $795c
a 00b23  20 5c 79 jsr $795c
a 00b26  4c 5c 79 jmp $795c
```

Aufruf mit
»SYS DEC("0B20"),0,0,0,0,,,« oder
»SYS DEC("0B20"),,,,,,,«.

Jeder andere Aufruf, der von dieser Form (Übergabe der Standardparameter und der einzulesenden Anzahl Kommata) abweicht, führt zu einem SYNTAX ERROR. Die Routine CHKKOM bietet daher eine hervorragende Absicherung gegen Fehleingaben beim Aufruf einer Assemblerroutine.

GETBYT ($87F4 ($87F1))

GETBYT verwendet auch der SYS-Befehl zur Übergabe der Standardparameter. Diese Routine holt einen Ein-Byte-Wert aus dem Basic-Text und übergibt ihn im X-Register. Liegt der gelesene Wert außerhalb der zulässigen Grenzen (0-255), wird ein ILLEGAL QUANTITY ERROR ausgegeben.

Vier Ein-Byte-Werte übergeben:

```
a 00b30  8d 00 0d sta $0d00
a 00b33  8e 01 0d stx $0d01
a 00b36  8c 02 0d sty $0d02
a 00b39  20 5c 79 jsr $795c
a 00b3c  20 f4 87 jsr $87f4
a 00b3f  00       brk
```

Nach dem Aufruf mit »SYS DEC("0B30"),1,2,3,0,4« werden die vier Standardparameter übergeben. Bevor wir nun die zusätzlichen Werte einlesen können, müssen wir zuvor die bereits übergebenen Werte retten. Im Beispiel werden die Registerinhalte nach $0D00 bis $0D02 übertragen. Anschließend wird mit CHKKOM das folgende Komma und mit GETBYT der zusätzliche Ein-Byte-Wert gelesen. Nach dem BRK sehen wir, daß das X-Register den Wert vier besitzt. Wenn Sie sich den Bereich ab $0D00 mit dem Monitor anschauen, werden Sie feststellen, daß der Reihe nach die Werte eins, zwei und drei abgelegt wurden.

Die Routine GETBYT besitzt zwei Einsprungadressen, $87F4 und $87F1. Bisher wurde der Einsprung ab Adresse $87F4 benutzt. Im Unterschied hierzu wird zu Beginn von $87F1 zuerst ein Unterprogrammaufruf von CHRGET durchgeführt, das heißt, das Zeichen, auf das der Programmzeiger weist, wird gelesen und inkrementiert.

Diese Einsprungadresse ist für »faule« Programmierer geeignet. Der Aufruf von CHRGET liest das Trennzeichen zwischen dem vorhergehenden Wert und dem einzulesenden Wert, so daß sich der Aufruf von CHKKOM erübrigt. Da dieses Trennzeichen jedoch in keiner Weise überprüft wird, verzichte ich persönlich auf diese Art des Einsprungs, da ich davon ausgehe, daß ein Benutzer, der sich beim Trennzeichen vertippt hat, eventuell auch den nachfolgenden Ein-Byte-Wert falsch eingegeben hat.



Für alle »Bytesparer« stelle ich dennoch ein kleines Beispielprogramm vor, das diesen Einsprung zum Lesen von fünf Parametern verwendet:

```
a 00b40  8d 00 0d sta $0d00
a 00b43  8e 01 0d stx $0d01
a 00b46  8c 02 0d sty $0d02
a 00b49  20 f1 87 jsr $87f1
a 00b4c  8e 03 0d stx $0d03
a 00b4f  20 f1 87 jsr $87f1
a 00b52  00       brk
```

Nach dem Aufruf mit »SYS DEC("0B40"),1,2,3,0,4,5« befindet sich im X-Register der zuletzt übergebene Wert fünf und die zuvor übergebenen Parameter in $0D00 bis $0D03.

Dieses Spiel kann beliebig weiter fortgesetzt werden. Der Zahl der Ein-Byte-Parameter, die an ein Assemblerprogramm übergeben werden können, sind keine Grenzen gesetzt. Sie müssen nur darauf achten, nach jeder Übergabe den übergebenen Wert zu retten, bevor der nächste Aufruf von CHKKOM oder GETBYT erfolgt.

Der nächste Schritt besteht darin, einen Zwei-Byte-Wert einzulesen. Eine solche Übergabe ist zum Beispiel für Grafikprogramme erforderlich, da die Auflösung des Grafikbildschirms in X-Richtung höher als 255 ist, oder für die Arbeit mit Arrays, bei denen das Basic-Programm der Routine übergeben soll, ab welchem Element (dessen Index höher als 255 sein kann) zum Beispiel eine Suche beginnen soll.

Die Übergabe eines Zwei-Byte-Wertes könnte noch vollzogen werden, indem das Basic-Programm vor dem Aufruf eine Zerlegung in Low- und Highbyte vornimmt.

Beispiel:

```
10 X=1030
20 H=INT(X/256)
30 L=X-H*256
40 SYS DEC("0B00"),L,H
```

Wie Sie an diesem Beispiel sehen, verarbeitet die Parameterübergabe beim SYS-Befehl auch Variablen und nicht nur direkt eingesetzte Zahlen. Problematisch bei dieser Form des Aufrufes ist jedoch, daß eine Assemblerroutine in einem Basic-Programm üblicherweise an zeitkritischen Stellen eingesetzt wird. Soll diese Routine nun zum Beispiel innerhalb einer Schleife ständig (mit variierenden Übergabeparametern) aufgerufen werden, leidet die Geschwindigkeit sehr unter dieser umständlichen Vorbereitung des Aufrufs.

Es ist daher vorteilhaft, auch beim Einlesen eines Zwei-Byte-Wertes die entsprechenden Routinen des Basic-Interpreters zu verwenden. Daß diese vorhanden sein müssen, beweist bereits der POKE-Befehl (POKE XXXX,YY), der ja ebenfalls einen Zwei-Byte-Wert als Parameter verwendet.

FRNUM ($77D7)

Die Routine FRNUM holt einen beliebigen numerischen Ausdruck aus dem Basic-Text und wertet ihn aus. Das Ergebnis wird im Fließkommaformat im Fließkommaakku Nummer 1 hinterlegt (FAC #1).

ADRFOR ($8815)

ADRFOR wandelt eine im FAC #1 abgelegte Fließkommazahl in das Adreßformat um, das heißt, in eine 16-Bit- oder Zwei-Byte-Zahl. Diese Adresse wird sowohl in den Registern (Y=Low-Byte/Akku=High-Byte) als auch an die Adressen $16, $17 ($16=Low-Byte/$17=High-Byte) übergeben.

Wenn wir beide Routinen nacheinander aufrufen (nachdem mit CHKKOM das Trennzeichen zur vorherigen Variablen gelesen wurde), erhalten wir sowohl in den Registern als auch in den genannten Speicherzellen einen Zwei-Byte-Wert:

```
a 00b60  20 5c 79 jsr $795c
a 00b63  20 d7 77 jsr $77d7
a 00b66  20 15 88 jsr $8815
a 00b69  00       brk
```

Rufen Sie diese Routine mit »SYS DEC("0B60"),0,0,0,0,1026« oder mit »SYS DEC("0B60"),,,,,1026« auf. Zuerst wird das Komma eingelesen, danach mit FRNUM der numerische Ausdruck geholt und im FAC #1 abgelegt, anschließend mit ADRFOR der FAC #1-Inhalt ins Adreßformat gewandelt. Im Akku befindet sich nun eine vier und im Y-Register der Wert zwei (hex: $0402; dezimal:1026). Die gleichen Werte finden Sie in $16, $17 in der Form: Lowbyte, Highbyte.

GETADR ($880F)

Die Routinen FRNUM und ADRFOR können Sie selbstverständlich auch einzeln für andere Zwecke als das Einlesen eines Zwei-Byte-Wertes einsetzen. Für diesen Zweck ist es empfehlenswerter, die Routine $GETADR einzusetzen. Diese Routine ruft nacheinander CHKKOM, FRNUM und ADRFOR auf. Das Einlesen des Zwei-Byte-Wertes aus dem vorherigen Beispiel vereinfacht sich dadurch erheblich. Beachten Sie, daß das Einlesen des Trennkommas mit CHKKOM entfällt, da auch diese Routine von GETADR aufgerufen wird:

```
a 00b70  20 0f 88 jsr $880f
a 00b73  00       brk
```

Rufen Sie diese Routine mit »SYS DEC("0B70"),0,0,0,0,1026« auf. Sie erhalten die gleichen Werte wie zuvor sowohl in den Registern als auch in den Speicherzellen $16, $17.

ADRBYT ($8803)

ADRBYT ruft nacheinander die Routinen GETADR, CHKKOM und GETBYT auf. Diese Routine liest daher in einem Zuge sowohl einen Zwei-Byte-Wert als auch einen folgenden Ein-Byte-Wert ein und ist damit hervorragend geeignet, um zum Beispiel Grafikroutinen aufzurufen und ihnen sowohl eine X-Koordinate (16-Bit) als auch eine Y-Koordinate (8-Bit) zu übergeben:

```
a 00b80  20 5c 79 jsr $795c
a 00b83  20 03 88 jsr $8803
a 00b86  00       brk
```

Aufruf:

»SYS DEC("0B80"),0,0,0,0,513,15«

Nach diesem Aufruf befindet sich im X-Register der von GETBYT übergebene Ein-Byte-Wert 15 (=hex $0F). Den Zwei-Byte-Wert 513 suchen Sie leider vergeblich im Y-Register und im Akku. Durch den nachfolgenden Aufruf von CHKKOM und GETBYT werden diese Register überschrieben, so daß diese Adresse diesmal nur in den Speicherzellen $16, $17 (Low/High) zu finden ist.

Da ich es wirklich beeindruckend finde, wieviel Vorbereitungen erspart werden können, indem man die für die jeweilige Aufgabe optimal geeignete Routine aufruft, stelle ich nun noch ein kleines Programm vor, das die Übergabe einer Adresse (16-Bit) und eines Bytes nur mit Hilfe der grundlegenden Routinen CHKKOM, FRNUM, ADRFOR und GETBYT vollzieht:

```
a 0b30 jsr $795c   ;CHKKOM
a 0b33 jsr $77d7   ;FRNUM
a 0b36 jsr $8815   ;ADRFOR
a 0b39 jsr $795c   ;CHKKOM
a 0b3c jsr $87f4   ;GETBYT
a 0b3f brk
```

Wenn Sie diese Serie von Aufrufen mit einem einzigen Aufruf von ADRBYT vergleichen, stellen Sie fest, daß es sich wirklich lohnt, über die verschiedenen Routinen zur Übergabe von numerischen Variablen Bescheid zu wissen, um die jeweils geeignetste einsetzen zu können.

GETPOS ($7AAF)

Zum Abschluß der verschiedenen Übergaberoutinen stelle ich nun noch GETPOS vor, die ganz sicher eine der wichtigsten Routinen des Basic-Interpreters ist. Die bisher vorgestellten Routinen bezogen sich nur auf numerische Varia-

blen und übergaben im Basic-Text stehende Werte und Werte numerischer Variablen.

Im Unterschied zu diesen Routinen kann GETPOS sowohl in Verbindung mit numerischen Variablen (Integer oder Fließkomma) als auch in Verbindung mit Stringvariablen eingesetzt werden. GETPOS holt keinen Wert, sondern liefert einen Zeiger auf eine Variable, genauer einen Zeiger auf die Variablentabelle.

Jede Variable ist in der Variablentabelle beschrieben, die normalerweise beim C 128 ab $0400 beginnt und in Bank 1 liegt, ebenso wie die Variablen selbst. Jeder Eintrag in dieser Tabelle ist bei nicht dimensionierten Variablen sieben Byte lang. Die ersten zwei Byte sind die beiden Zeichen des Variablennamens, wobei bei Integerzahlen in beiden Byte Bit sieben, bei Stringvariablen Bit sieben nur im zweiten Byte des Namens gesetzt ist, und Realzahlen durch ein in beiden Bytes gelöschtes Bit sieben erkannt werden. Die restlichen fünf Byte (3 bis 7) haben folgende Bedeutung:
a) Real: (5 Byte Fließkommadarstellung)
b) Int: (High-Byte)(Low-Byte)(3 Byte ungenutzt)
c) String:(1 Byte Stringlänge)(2 Byte Stringadresse)(2 Byte ungenutzt)

Beachten Sie unbedingt, daß eine Integerzahl im – für Assemblerprogrammierer ungewohnten – Format: »High-Byte/Low-Byte« abgelegt wird.

Dimensionierte Arrays werden folgendermaßen verwaltet: Der Variablenname (2 Byte) befindet sich nur am Arraybeginn. Danach folgen zwei Byte, die den belegten Speicherplatz angeben, ein weiteres Byte, das die Dimensionierung enthält und zwei Byte mit der Anzahl der Arrayelemente. Erst anschließend folgt das Array:
a) Real: (5 Byte Fließkommadarstellung)
b) Int: (High-Byte)(Low-Byte)
c) String:(1 Byte Stringlänge)(2 Byte Stringadresse)

GETPOS liefert in jedem dieser Fälle einen Zeiger auf das erste Byte des Descriptors der jeweiligen Variablen, das heißt, bei nicht dimensionierten Variablen einen Zeiger auf das erste Byte des Variablennamens, bei dimensionierten Variablen entweder – numerische Variablen – einen Zeiger auf das erste Byte der Zahl selbst, oder aber – Stringvariablen – einen Zeiger auf die Länge der jeweiligen Stringvariablen.

Dieser Zeiger wird sowohl an die Register (Akku = Low/Y = High), als auch an die Speicherzellen $49, $4A (Low-/Highbyte) übergeben.

Da GETPOS alle Variablentypen verarbeitet, genügt ein winziges Beispielprogramm, um den Zugriff auf numerische und Stringvariablen, sowohl dimensioniert als auch nicht dimensioniert, zu demonstrieren:

```
a 00b90  20 5c 79 jsr $795c
a 00b93  20 af 7a jsr $7aaf
a 00b96  00       brk
```

Die folgenden Beispiele können Sie im Direktmodus eingeben:

a) nicht dimensionierte Variablen

1.NEW:A% = 10:SYS DEC("0B90"),0,0,0,0,A%

Gehen Sie in den Monitor und geben Sie ein:

M 49. Ergebnis: 00049 02 20 ......

Der Zeiger $49, $4A weist somit auf $2002 (in Bank 1, der Variablenbank). Geben Sie ein:

M 12002. Ergebnis: 12002 00 0A ......

In $12002 befindet sich das High-Byte (Null), in $12003 das Lowbyte ($0A = 10) der Integerzahl 10.

2.NEW:A = 10:SYS DEC("0B90"),0,0,0,0,A

M 49. Ergebnis: 00049 02 20 ......

M 12002. Ergebnis: 12002 84 20 00 00 00 (= Fließkommadarstellung von 10).

3.NEW:A$ = "BASICINTERPRETER":SYS DEC("0B90"),0,0,0,0,A$

M 49. Ergebnis: 00049 02 20 ......

M 12002. Ergebnis: 12002 10 EE FE ......

Im Unterschied zu numerischen Variablen zeigt der Zeiger $49, $4A bei Stringvariablen nicht unmittelbar auf die Variable selbst, sondern auf das erste Descriptorbyte (Stringlänge). Die beiden folgenden Bytes geben die Adresse des Strings an. Geben Sie daher ein:

M 1FEEE. Ergebnis: 1FEEE 42 41 53 ..... (= ASCII-Darstellung von 'BASICINTERPRETER').

b) dimensionierte Variablen

1.NEW:A%(2)= 10:SYS DEC("0B90"),0,0,0,0,A%(2)

M 49. Ergebnis: 00049 0B 20......

M 1200B. Ergebnis:
1200B 00 0A......

In $12002 befindet sich das High-Byte (Null), in $12003 das Lowbyte ($0A = 10) der Integerzahl 10.

2.NEW:A(2) = 10:SYS DEC("0B90"),0,0,0,0,A(2)

M 49. Ergebnis: 00049 11 20......

M 12011. Ergebnis: 12011 84 20 00 00 00 (= Fließkommadarstellung von 10).

3.NEW:A$(2) = "C 128":SYS DEC("0B90"),0,0,0,0,A$(2)

M 49. Ergebnis: 00049 0D 20......

M 1200D. Ergebnis:
1200D 05 F9 FE.......

M 1FEF9.Ergebnis:
1FEF9 43 2D 31.....
(= ASCII-Darstellung von »C 128«)

Wie Sie sehen, ist diese Routine außerordentlich universell einsetzbar. Merken Sie sich folgende Punkte, die sowohl für einzelne als auch für Arrayvariablen gelten: Der Zeiger in den Speicherzellen $49, $4A zeigt bei numerischen Variablen auf das erste Byte der Zahl selbst, bei Stringvariablen auf den Längendescriptor des Strings, hinter dem in Form von Lowbyte/Highbyte die Stringadresse folgt.

Weiterhin ist eventuell von Interesse für Sie, daß beim Aufruf dieser Routine in den Speicherzellen $47, $48 der Variablenname übergeben wird und daß GETPOS eine Variable anlegt, wenn sie noch nicht definiert wurde. Daher besteht keine Veranlassung, darauf zu achten, daß beim Aufruf einer eigenen Assemblerroutine, die GETPOS verwendet wird. Ein Absturz ist dank des Anlegens der Variablen nicht möglich!

Einen Nachteil besitzt dagegen diese Routine gegenüber ihrem Äquivalent auf dem C 64: GETPOS liefert auf dem C 64 einen zweiten Zeiger auf den Anfang eines Arrays bei Verwendung dimensionierter Variablen. Dieser Zeiger kann zum Beispiel bei Suchroutinen sehr nützlich sein, die ein Arrayfeld nach einem bestimmten Wert oder String durchsuchen müssen. Eine solche Routine muß wissen, wann das Arrayende erreicht wird und die Suche beendet werden muß. Dieses Arrayende kann über den erwähnten Zeiger ermittelt werden, indem der Arraydescriptor (Speicherplatz, den das Array benötigt), der in der Arraybeschreibung steht, zum Arraybeginn addiert wird (= Ende des Arrays).

GETPOS auf dem C 128 besitzt diesen Zeiger zwar ebenfalls, er wird jedoch vor der Rückkehr aus GETPOS durch andere Werte überschrieben. Um dennoch auf die Descriptoren eines Arrays zuzugreifen, kann ein Trick verwendet werden. Beispiel: Eine Routine soll ein Stringarray ab Element A$(53) durchsuchen. Das Arrayende kann durch folgende Form des Aufrufs ermittelt werden:

SYS DEC("0BA0"),0,0,0,0,A$(0),A$(53)

Die aufgerufene Assemblerroutine lautet:

```
a 00ba0  20 5c 79 jsr $795c
;CHKKOM
a 00ba3  20 af 7a jsr $7aaf
;GETPOS von A$(0)
a 00ba6  38       sec
a 00ba7  e9 07    sbc #$07
;7 von Low-Byte subtrahieren
a 00ba9  85 fb    sta $fb
;Low-Byte speichern
a 00bab  b0 01    bcs $0bae
;Unterlauf?
a 00bad  88       dey
;High-Byte dekrementieren
a 00bae  84 fc    sty $fc
;High-Byte speichern
a 00bb0  20 5c 79 jsr $795c
;CHKKOM
a 00bb3  20 af 7a jsr $7aaf
;GETPOS von A$(53)
a 00bb6  00       brk
```

Diese Routine holt zuerst die Adresse der Descriptoren des ersten Arrayelementes, A$(0). Der Zeiger $49, $4A weist somit auf den Längendescriptor dieses ersten Elementes. Wird nun von diesem Zeiger – der sich zugleich in den Registern befindet (Akku = Lowbyte/Y-Reg. = High-Byte) – der Wert sieben subtrahiert, weist der Zeiger auf das erste Byte der Arraybeschreibung, die wie erwähnt sieben Byte lang ist.

Dieser neue Zeiger wird in $FB, $FC gespeichert und anschließend GETPOS auf das eigentlich interessierende Startelement A$(53) angewendet, um dessen Länge und Adresse zu holen.

Da $FB, $FC nun auf das erste Element der Arraybeschreibung weist, kann durch Addition des vom Array benötigten Speicherplatzes zum Arrayanfang dessen Ende +1, das heißt, der Beginn des nächsten Arrays, ermittelt werden. Da die Speicherbelegung in Byte zwei und drei der Arraybeschreibung enthalten ist (Low-Byte/High-Byte), kann mit indirekt indizierter Adressierung darauf zugegriffen werden.

Prinzip:

```
LDY #2
LDA ($FB),Y   ;Zugriff auf
    Low-Byte des benötigten
    Speicherplatzes
INY
LDA ($FB),Y   ;Zugriff auf High-
    Byte des benötigten
    Speicherplatzes
```

Wir kommen nun zum Anlegen von Variablen aller Art. Zum Anlegen müssen wir jedoch auf die Variablentabelle und – bei Stringvariablen – auf den Stringstack zugreifen. Sollten Sie anderer Meinung sein, da die Variablentabelle normalerweise ab $0400 beginnt und der Bereich $0000 bis $1FFF von unserem Initialisierungsprogramm als gemeinsamer Bereich von Bank 0 und Bank 1 definiert wurde, so schauen Sie sich bitte mit dem Monitor die Speicherzellen $2F,$2E an, den Zeiger auf den Anfang der Variablentabelle. Dieser Zeiger weist auf $2000, ein Ergebnis des Basic-Initialisierungs-Programms. Damit liegt der Variablenbereich außerhalb des gemeinsamen Bereichs. Beim Zugriff auf die Tabelle muß zuvor auf Bank 1 umgeschaltet werden.

Mit dem eingebauten Monitor ist ein solcher Zugriff sehr einfach, da vor der eigentlichen Adresse nur eine Eins (Bank 1) angegeben werden muß und der Monitor dann selbsttätig auf die gewünschte Bank schaltet. In unseren eigenen Programmen müssen wir jedoch »per Hand« zwischen Bank 0 und Bank 1 umschalten. Ich stelle nun ein erweitertes Initialisierungs-Programm vor, in dem zwei Umschaltroutinen vorhanden sind:

```
a 00b00  ad 00 ff lda $ff00
a 00b03  48       pha
a 00b04  a9 00    lda #$00
a 00b06  8d 00 ff sta $ff00
a 00b09  ad 06 d5 lda $d506
a 00b0c  09 06    ora #$06
a 00b0e  8d 06 d5 sta $d506
a 00b11  68       pla
a 00b12  8d 00 ff sta $ff00
a 00b15  60       rts
a 00b16  08       php
a 00b17  48       pha
a 00b18  a9 00    lda #$00
a 00b1a  8d 00 ff sta $ff00
a 00b1d  68       pla
a 00b1e  28       plp
a 00b1f  60       rts
a 00b20  08       php
a 00b21  48       pha
a 00b22  a9 7f    lda #$7f
a 00b24  8d 00 ff sta $ff00
a 00b27  68       pla
a 00b28  28       plp
a 00b29  60       rts
```

Dieses erweiterte Initialisierungs-Programm enthält eine Routine zum Umschalten auf Bank 0 ab $0B16 und eine zweite zum Umschalten auf Bank 1 ab $0B20. In beiden Fällen wird der Inhalt der beiden Register, die beim Ablauf verändert werden, Akku und Status, gerettet und nach beendetem Umschalten zurückgeholt. Beide Routinen können Sie daher in Ihren eigenen Programmen an beliebigen Stellen aufrufen, ohne sich Gedanken darüber zu machen, welche momentan benötigten Registerinhalte verändert werden könnten und daher vor dem Aufruf gerettet werden müssen.

Wichtig ist dieses Retten der Register dann, wenn Sie zum Beispiel einen Wert aus der Variablentabelle holen wollen:

```
jsr bank 1
lda vartab,x
cmp vartab,y
jsr bank 0
```

Nach dem Zurückschalten auf Bank 0 sind alle Flaggen, die der CMP-Befehl gesetzt hat, unverändert vorhanden.

Integervariable anlegen

Zum Anlegen einer Integervariablen genügen die bereits besprochenen Routinen des Basic-Interpreters. Wir gehen davon aus, daß unser Programm zum Beispiel nach dem Durchsuchen eines Arrays den Index eines Elementes, das dem von Basic übergebenen Suchkriterium entspricht, in einer Variablen an das Basic zurückübergeben soll:

```
a 00b30  20 5c 79 jsr $795c
a 00b33  20 af 7a jsr $7aaf
a 00b36  20 20 0b jsr $0b20
a 00b39  a0 00    ldy #$00
a 00b3b  a9 02    lda #$02
a 00b3d  91 49    sta ($49),y
a 00b3f  c8       iny
a 00b40  a9 04    lda #$04
a 00b42  91 49    sta ($49),y
a 00b44  20 16 0b jsr $0b16
a 00b47  60       rts
```

Rufen Sie dieses Programm mit »SYS DEC("0B30"),0,0,0,0,I%« auf. Denken Sie jedoch daran, zuvor das erweiterte Initialisierungs-Programm einzugeben!

Unsere Routine liest nun das Komma ein und ruft GETPOS auf, die in den Speicherzellen $49, $4A einen Zeiger auf die – neu angelegte – Variable liefert, das heißt, auf das erste Byte der Integerzahl selbst, die momentan natürlich Null ist. Nachdem auf Bank 1, die Variablenbank, umgeschaltet wurde, schreiben wir in das erste Byte dieser Zahl eine $02, in das zweite Byte den Wert $04. Zum Abschluß schalten wir wieder auf Bank 0 um.

Geben Sie bitte »PRINT I%« ein, nachdem die Routine bearbeitet wurde. Jedem, der nun erwartet, daß der Wert 1026 ausgegeben wird, empfehle ich, den kleinen Exkurs über die Variablenabspeicherung nachzulesen. Integervariablen werden in der ungewohnten Form »High/Low« abgelegt. Die als Wert eingetragenen Zahlen $02, $04 entsprechen somit dem Wert

$0204 beziehungsweise der ausgegebenen Dezimalzahl 516.

Anlegen einer Realvariablen

Wenn wir einem aufrufenden Basic-Programm ein Realergebnis mitteilen wollen, legen wir auch diese Realvariable analog dem eben geschilderten Ablauf an: Wir holen die Variablenadresse, schalten auf Bank 1 um, legen die Realzahl im Fließkommaformat ab der angezeigten Adresse ab und schalten wieder auf Bank 0:

```
a 00b50  20 5c 79 jsr $795c
a 00b53  20 af 7a jsr $7aaf
a 00b56  20 20 0b jsr $0b20
a 00b59  a0 00    ldy #$00
a 00b5b  a9 84    lda #$84
a 00b5d  91 49    sta ($49),y
a 00b5f  c8       iny
a 00b60  a9 20    lda #$20
a 00b62  91 49    sta ($49),y
a 00b64  a9 00    lda #$00
a 00b66  c8       iny
a 00b67  91 49    sta ($49),y
a 00b69  c8       iny
a 00b6a  91 49    sta ($49),y
a 00b6c  c8       iny
a 00b6d  91 49    sta ($49),y
a 00b6f  20 16 0b jsr $0b16
a 00b72  60       rts
```

Starten Sie dieses Programm mit »SYS DEC("0B50"),0,0,0,0,R«. Geben Sie anschließend ein: »PRINT R«. Wir erhalten den Dezimalwert 10 oder in Fließkommadarstellung: »84 20 00 00 00«.

Wie Sie sehen, ist es völlig unproblematisch, numerische Variablen auf dem C 128 anzulegen. Der entscheidende Punkt ist das Umschalten auf Bank 1, bevor auf die Variablentabelle zugegriffen wird. Im Unterschied zum C 64 sollten Sie bei der Fehlersuche in Ihren Programmen daher zuerst nachforschen, ob diese Umschaltung an den nötigen Stellen vorgenommen wird, da ansonsten auch das durchdachteste Programm auf dem C 128 nicht laufen wird.

Die Datentyp-Flags ($0F/$10)

Bevor wir uns nun dem Anlegen von Strings zuwenden, will ich zuvor zwei wichtige Zeropageadressen erwähnen, $0F und $10. Wenn wir die Routine GETPOS aufrufen, erhalten wir in diesen beiden Speicherzellen Informationen über die Art der gelesenen Variablen.

Wie es bereits im Handbuch steht, enthält $0F den Wert $00 bei numerischen Variablen und $80 bei Stringvariablen.

$10 enthält im Falle einer Integervariablen den Wert $80 und $00 im Fall einer Realvariablen.

Wie wir beide Speicherzellen sinnvoll verwenden können, will ich an einem Beispiel demonstrieren, an einer kleinen Routine, die die gleiche Funktion erfüllt wie der Befehl SWAP in manch anderem Basic-Interpreter, nämlich die Vertauschung zweier Variablen.

Ein solcher Befehl ist vor allem in Sortierroutinen, in denen ständig Variablen miteinander vertauscht werden müssen, äußerst nützlich. Der Befehl:

SWAP A$(2),A$(7) ist zum einen schneller als die herkömmliche Methode:

S$=A$(2):A$(2)=A$(7):A$(7)=S$

und der zweite, fast noch wesentlichere Vorteil besteht darin, daß bei der Vertauschung von Stringvariablen die Strings nicht am unteren Ende des Stringstacks neu angelegt werden müssen, sondern daß die Vertauschung der Stringdescriptoren genügt.

Die Arbeit einer Sortierroutine, die einen derartigen Befehl verwendet, wird daher weitaus seltener durch eine nötige Garbage Collection unterbrochen, so daß sich die Sortierzeit erheblich verkürzt.

Bevor Sie die folgenden Programmzeilen eingeben, vergewissern Sie sich bitte, daß sich das neuere Initialisierungs-Programm im Speicher befindet. Unsere Routine wird die darin enthaltenen Unterprogramme zum Umschalten der Speicherbänke benötigen.

Zeiger auf die Variablen holen

```
a 00b30  20 5c 79 jsr $795c
a 00b33  20 af 7a jsr $7aaf
a 00b36  a5 49    lda $49
a 00b38  a6 4a    ldx $4a
a 00b3a  85 fb    sta $fb
a 00b3c  86 fc    stx $fc
a 00b3e  20 5c 79 jsr $795c
a 00b41  20 af 7a jsr $7aaf
```

Der erste Programmteil bietet nur bereits Bekanntes. Nach dem Aufruf mit zum Beispiel »SYS DEC("0B30"),0,0,0,0,A$,B$« wird ein Zeiger auf A$ mit Hilfe von GETPOS geholt. Dieser Zeiger wird nach $FB, $FC kopiert, da anschließend ein Zeiger auf B$ geholt wird und sich in den Speicherzellen $49, $4A nun der Zeiger auf die Descriptoren dieser Variablen befinden.

Typflags auswerten

Die Auswertung verläuft nach dem abgebildeten Schema: Wenn $0F gleich Null ist, handelt es sich um eine numerische Variable und es wird geprüft, ob $10 ebenfalls den Wert Null enthält. Wenn ja, wurde eine Realvariable gelesen und wir laden das Y-Register mit $04, dem Startwert für eine Schleife, die von Y bis inklusive Null läuft und die einzelnen Bytes beider Variablen (beziehungsweise bei Strings deren Descriptoren) vertauscht.

War $0F gleich Null, $10 jedoch ungleich Null, wird das Y-Register mit $01 geladen (Integervariable).

Wurde hingegen eine Stringvariable gelesen, laden wir das Y-Register mit dem Wert $02.

```
        lda $0f
        bne string
        lda $10
        bne integer
        ldy #$04
        bne weiter
integer ldy #$01
        bne weiter
string  ldy #$02
weiter  jsr $0b20
```

Soweit der prinzipielle Ablauf, den ich jedoch in ein kürzeres Programm umgesetzt habe:

```
a 00b44  a5 0f    lda $0f
a 00b46  d0 0a    bne $0b52
a 00b48  a5 10    lda $10
a 00b4a  d0 03    bne $0b4f
a 00b4c  a0 04    ldy #$04
a 00b4e  2c a0 01 bit $01a0
a 00b51  2c a0 02 bit $02a0
a 00b54  20 20 0b jsr $0b20
```

Dieser Programmteil wird für erfahrene 6502/8502 – Programmierer auf Anhieb verständlich sein, für weniger erfahrene jedoch wahrscheinlich ein unlösbares Rätsel darstellen.

Zuerst schauen wir uns den Inhalt des Flags $0F an. Wenn es gleich Null ist, das heißt eine numerische Variable übergeben wird, wird $10 getestet. Handelte es sich um eine Realvariable, wird das Y-Register unmittelbar mit $04 geladen.

Das auf den Befehl LDY #04 folgende Byte $2C ist der Prozessorcode für den BIT-Befehl mit 16-Bit-Adressierung. Die nachfolgenden beiden Bytes werden somit als Adresse angesehen.

Der BIT-Befehl hat keinerlei Auswirkungen auf das Y-Register. Da anschließend wiederum Byte $2C folgt, werden auch die folgenden beiden Bytes als Adresse eines BIT-Befehls interpretiert.

Seltsamerweise springt unser Programm mitten in den BIT-Befehl »hinein«, wenn eine Integervariable übergeben wurde (a 00b4a d0 03 bne $0b4f). Geben Sie bitte mit dem Monitor ein:

d 0b4f. Er interpretiert die auf den BIT-Code folgenden Bytes als: »LDY #$01«, wodurch das Y-Register – wie im Falle einer Integervariablen gewünscht – mit einer Eins geladen wird.

Soviel zu den Einsatzmöglichkeiten des BIT-Befehls. Daß diese



Anwendung üblich ist, zeigt der ständige Einsatz dieser Methode im ROM des C 128.

Vertauschen der Variablen (-zeiger)

```
a 00b57  b1 49     lda ($49),y
a 00b59  aa        tax
a 00b5a  b1 fb     lda ($fb),y
a 00b5c  91 49     sta ($49),y
a 00b5e  8a        txa
a 00b5f  91 fb     sta ($fb),y
a 00b61  88        dey
a 00b62  10 f3     bpl $0b57
a 00b64  20 16 0b  jsr $0b16
a 00b67  60        rts
```

Das Vertauschen der beiden Variablen geschieht analog zu den erwähnten Basic-Befehlen, jedoch byteweise. Als Zwischenspeicher dient das X-Register.

Beachten Sie, daß die Routine nicht überprüft, ob beide übergebenen Variablen vom gleichen Typ sind, was Grundbedingung für eine sinnvolle Vertauschung ist.

Komplettes Listing von »SWAP«

```
a 00b30  20 5c 79  jsr $795c
a 00b33  20 af 7a  jsr $7aaf
a 00b36  a5 49     lda $49
a 00b38  a6 4a     ldx $4a
a 00b3a  85 fb     sta $fb
a 00b3c  86 fc     stx $fc
a 00b3e  20 5c 79  jsr $795c
a 00b41  20 af 7a  jsr $7aaf
a 00b44  a5 0f     lda $0f
a 00b46  d0 0a     bne $0b52
a 00b48  a5 10     lda $10
a 00b4a  d0 03     bne $0b4f
a 00b4c  a0 04     ldy #$04
a 00b4e  2c a0 01  bit $01a0
a 00b51  2c a0 02  bit $02a0
a 00b54  20 20 0b  jsr $0b20
a 00b57  b1 49     lda ($49),y
a 00b59  aa        tax
a 00b5a  b1 fb     lda ($fb),y
a 00b5c  91 49     sta ($49),y
a 00b5e  8a        txa
a 00b5f  91 fb     sta ($fb),y
a 00b61  88        dey
a 00b62  10 f3     bpl $0b57
a 00b64  20 16 0b  jsr $0b16
a 00b67  60        rts
```

Zum Testen des Programms geben Sie bitte ein:

```
A$="C 128":B$="C 64"
SYS DEC("0B30"),,,,,A$,B$
PRINT A$,B$
```

und Sie erhalten auf dem Bildschirm die vertauschten Stringinhalte:

C 64 C 128

Anlegen einer Stringvariablen

Die bisherigen Programmbeispiele dieses Artikels wurden so einfach wie möglich gehalten und dürften keine Verständnisschwierigkeiten hervorrufen. Das Anlegen einer Stringvariablen werde ich nun an einem komplexeren Beispielprogramm erläutern, einer INPUT-Routine.

Bei der Entwicklung dieser Routine werden viele der beschriebenen Interpreter-Routinen benötigt. Außerdem werden weitere Routinen und spezielle Zeropageadressen beschrieben, ohne die das Anlegen eines Strings nicht möglich ist.

Das Niveau dieses Artikels wird durch das komplexere Programm zwar deutlich höher werden, dafür haben Sie jedoch am Ende dieses Abschnitts eine lauffähige Eingaberoutine in Ihren Händen, die Sie jederzeit ausbauen können und kennen das Anlegen von Strings auf dem C 128, das in den verschiedensten Routinen von Bedeutung ist. Einsatzbeispiel sind wie erwähnt Sortierroutinen, INPUT-Routinen; eine Vielzahl weiterer Routinen, die Strings bearbeiten, sind denkbar. Eine nette Aufgabe könnte zum Beispiel eine INPUT#-Routine sein, die Strings mit bis zu 255 beliebigen Zeichen einliest und damit die Beschränkungen des INPUT#-Befehls aufhebt. Die INPUT-Routine soll folgendes leisten:

1. Vor Beginn der Eingabe den Cursor auf eine angegebene Position setzen.
2. Während der Eingabe nur Zeichen zulassen, die in einem als Parameter übergebenen String enthalten sind.
3. Prinzipiell - soweit nicht durch Punkt 2 verhindert - alle Zeichen zulassen, auch solche, die beim INPUT-Befehl zu einem »REDO FROM START«-Error führen.
4. Die Eingabe vom Bildschirm übernehmen und in einem vom Basic-Programm frei wählbaren String an dieses übergeben.

Da als Eingabezeichen nur die in dem übergebenen String enthaltenen zugelassen werden sollen, ist es nötig, diesen String um:

CHR$(29) = Cusor-right,
CHR$(157) = Cursor-left,
CHR$(20) = Delete und
CHR$(148) = Insert zu erweitern,

um die gleichen Editiermöglichkeiten wie beim INPUT-Befehl zu erhalten.

Beispiel:

```
Z$="1234567890"+CHR$(29)+
CHR$(157)+CHR$(20)+CHR$(148)
```

läßt Ziffern und Editiertasten als Eingabe zu.

Da dieses Programm deutlich umfangreicher und damit schwieriger zu verstehen ist als die bisherigen Programmbeispiele, werde ich den Programmablauf diesmal anhand eines Assemblerlistings erklären. Zum Abtippen mit dem eingebauten Monitor finden Sie im Anschluß an die Erläuterungen das Programm in der gewohnten disassemblierten Form.

Zeropageadressen

```
**ZEROPAGEADRESSEN*
STREND = $35    ;ZEIGER AUF ENDE
                 DES STRINGSTACKS
ZEIGER = $49    ;ZEIGER AUF
                 DESCRIPTOREN
SPALTE = $EC    ;CURSORSPALTE
ZEILPOI = $E0   ;ZEIGER:SPALTE 0
                 DER MOMENTANEN
                 CURSORZEILE
```

STREND zeigt immer das momentane Ende des Stringstacks, der bekanntlich von »oben« ($FFFF) nach unten wächst. ZEIGER ist der von GETPOS übergebene Zeiger auf die Descriptoren einer aus dem Basic-Text gelesenen Variablen. In SPALTE ist – bei Benutzung der Betriebssystem-Routinen zur Zeichenausgabe – die momentane Cursorspalte enthalten, und ZEIGER weist auf den Anfang der aktuellen Zeile, in der sich der Cursor befindet. Die aktuelle Cursorposition ergibt sich somit aus ZEIGER + SPALTE.

Eigene Variablen

```
**EIGENE VARIABLEN**
BANK0    = $0B16
BANK1    = $0B20
ZLAENGE  = $FB  ;LAENGE DES
                 STRINGS MIT DEN
                 ZUGEL.ZEICHEN
POSIT    = $FC  ;ZEIGER:POS. DES
                 STRINGS MIT DEN
                 ZUGEL.ZEICHEN
STARTPOS = $A3  ;ZEIGER:
                 STARTPOSITION DER
                 EINGABE
LAENGE   = $A5  ;EINGABEL.MAX.
```

BANK0 und BANK1 sind die Einsprungadressen für unsere Unterprogramme zur Bankumschaltung. In ZLAENGE und in POSIT, POSIT+1 werden die Descriptoren des Strings gespeichert, in dem die als Eingabe zulässigen Zeichen abgelegt wurden. STARTPOS, STARTPOS+1 ist die beim Aufruf übergebene Bildschirmposition, bei der die Eingabe beginnen soll. In LAENGE legen wir die ebenfalls vom Basic-Programm übergebene maximale Eingabelänge ab.

Verwendete Kernelroutinen

Zu diesen Routinen dürfte keine weitere Erläuterung nötig sein, da sie den entsprechenden C 64-Rou-

tinen äquivalent sind und sich dank der Sprungtabelle auch die Einsprungadressen nicht verändern.

```
**KERNELROUTINEN**
BSOUT = $FFD2    ;ZEICHEN IM AKKU
                  AUSGEBEN
BASIN = $FFCF    ;ZEICHEN AUS
LOG. DATEI HOLEN
CHKKOM = $795C   ;AUF KOMMA
                  PRUEFEN
GETPOS = $7AAF   ;DESCRIPTOREN
                  POS. EINER
                  VARIABLEN
                  HOLEN
STRRES = $9299   ;SPEICHERPLATZ
                  RESERVIEREN
GETIN = $FFE4    ;ZEICHEN HOLEN
SETCRS = $FFF0   ;CURSOR SETZEN
OPEN = $FFC0     ;DATEI OEFFNEN
CLOSE = $FFC3    ;DATEI
                  SCHLIESSEN
CHKIN = $FFC6    ;EINGABE AUF
                  LOG. DATEI LEGEN
CLRCH = $FFCC    ;EINGABE AUF
                  TASTATUR LEGEN
SETFLS = $FFBA   ;FILEPARAMETER
                  SETZEN
SETNAM = $FFBD   ;FILENAME SETZEN
 .BA $0D00       ;PROGRAMM-
                  START
```

Parameter retten/Cursor setzen

Aufgerufen werden soll unsere Routine mit:

SYS DEC("0D00"),Länge,Zeile,Spalte,0,Zeichenstring,Übergabestring

Beispiel:

SYS DEC("0D00"),20,10,5,0,Z$,A$

bedeutet: die maximale Eingabelänge beträgt 20 Zeichen, die Eingabe soll ab Spalte 5 in Zeile 10 beginnen, nur die in Z$ enthaltenen Zeichen werden als Eingabe zugelassen, und die Eingabe selbst soll dem Basic-Programm in A$ übergeben werden (vergessen Sie bei der Definition von Z$ die Editiertasten nicht).

Die in den Registern übergebenen Parameter Länge/Zeile/Spalte müssen für eine spätere Verwendung gerettet werden:

```
**PARAMETER RETTEN*
STA LAENGE
TXA
PHA
TYA
PHA
```

Der Cursor wird nun auf die Zeile und Spalte des Eingabebeginns gesetzt (X=Zeile/Y=Spalte):

```
**CURSOR SETZEN*
CLC
JSR SETCRS ;CURSOR SETZEN
```

Descriptoren holen

Der nächste Schritt besteht darin, die Descriptoren für den String mit den als Eingabe zulässigen Zeichen zu holen:

```
**ZEIGER:DESCR.ZEICHEN$ HOLEN**
        JSR CHKKOM
        JSR GETPOS
        JSR BANK1
        LDY #2
HOLDES  LDA (ZEIGER),Y
        STA ZLAENGE,Y
        DEY
        BPL HOLDES
        JSR BANK0
```

Zuerst wird das Komma hinter den Standardparametern gelesen, anschließend GETPOS aufgerufen, die uns in den Speicherzellen $49, $4A (=ZEIGER) einen Zeiger auf den nachfolgenden String (im Beispielaufruf Z$) liefert beziehungsweise diesen String zugleich anlegt, wenn er noch nicht existiert.

Wir lesen nacheinander alle drei Descriptor-Bytes (Stringlänge; Low- und High-Byte der Stringposition) und übertragen sie nach ZLAENGE und ZLAENGE, ZLAENGE+1 beziehungsweise POSIT und POSIT+1 (ZLAENGE+1=POSIT und ZLAENGE+2=POSIT+1, siehe Deklaration eigener Variablen).

Achten Sie bitte darauf, daß diese Routine unbedingt die C 128-spezifische Eigenschaft des Bankswitching berücksichtigen muß! Beim Lesen der Descriptoren greifen wir auf die Variablentabelle in Bank 1 und müssen daher unbedingt BANK1 aufrufen, unser Unterprogramm zum Umschalten auf diese Bank. Da nach der Rückkehr aus GETPOS Bank 0 eingeschaltet ist, würden wir sonst anstelle der Variablentabelle die Bytes an den entsprechenden Adressen in Bank 0 lesen. Nach dem Beenden der Routine schalten wir wieder auf die Standardbank 0 zurück.

Eingabeschleife/Zeichen unter Cursor invertieren

Die Aufgabe der Eingabeschleife ist schnell erklärt: Sie ruft GETIN auf, eine Routine, die analog dem Basic-Befehl GET ist, das heißt, ein Zeichen von der Tastatur liest, jedoch nicht wartet, wenn keine Taste gedrückt ist. Das Zeichen wird im Akku übergeben. Die Eingabeschleife wird erst verlassen, wenn der Akkuinhalt ungleich Null ist, das heißt, wenn eine Taste gedrückt wurde.

```
**EINGABESCHLEIFE**
GET  JSR INVERT  ;ZEICHEN INVERT.
GET1 JSR GETIN
     BEQ GET
     JSR INVERT
```

INVERT stellt ein Unterprogramm dar, das das Zeichen an der momentanen Cursorposition invertiert. Vor einer Eingabe wird das jeweilige Zeichen durch den Aufruf von INVERT daher revers dargestellt, nach einer Eingabe wieder invertiert, das heißt normal dargestellt. Diese Routine ist nötig, da GETIN ebenso wie GET keinen blinkenden Cursor erzeugt. Durch INVERT ergibt sich zwar ebenfalls kein blinkender, jedoch zumindest ein stehender Cursor, der dem Benutzer die momentane Eingabeposition angibt.

```
**Z.UNTER CRS.INVERTIEREN**
INVERT PHA
       LDY SPALTE
       LDA (ZEILPOI),Y
       EOR #128
       STA (ZEILPOI),Y
       PLA
       RTS
```

Beachten Sie bitte, daß das im Akku übergebene Zeichen durch INVERT nicht verändert werden darf, und daher auf den Stack gerettet und vor dem Rücksprung wieder vom Stack geholt wird. INVERT folgt im Programm übrigens nicht der Eingabeschleife, sondern befindet sich am Programmende.

Zeichenüberprüfung

Nach der Eingabe eines Zeichens muß überprüft werden, ob es sich um Carriage Return (RETURN = Code $0D) handelt. Wenn ja, ist die Eingabe beendet und der String mit den eingegebenen Zeichen wird angelegt (Routine RETURN).

```
**RETURN?**
CMP #$0D
BEQ RETURN
```

Wurde nicht RETURN gedrückt, muß überprüft werden, ob ein im Zeichenstring (im Beispiel Z$) enthaltenes Zeichen eingegeben wurde.

```
**ZULAESSIGES ZEICHEN?**
        LDY ZLAENGE
        DEY
COMPARE JSR BANK1
        CMP (ZPOSIT),Y
        JSR BANK0
        BEQ AUSGABE
        DEY
        BPL COMPARE
        BMI GET     ;ABS.SPRUNG
```

Erinnern Sie sich: in ZLAENGE wurde die Länge des Zeichenstrings und in ZPOSIT, ZPOSIT+1 der Descriptorzeiger auf den String selbst übertragen. Zur Überprüfung greifen wir in einer Schleife auf jedes einzelne Zeichen dieses Strings zu und vergleichen es mit dem im Akku enthaltenen Eingabe-



zeichen. Vor jedem Vergleich schalten wir auf Bank 1, also auf die Bank, in der Variablentabelle und Stringstack enthalten sind, und anschließend auf Bank 0. Diese Lösung ist sicher nicht die schnellste, die Geschwindigkeit der vorgestellten INPUT-Routine ist jedoch auch für »Schnelltipper« bei weitem ausreichend.

Diese bereits einmal angewendete Lösung für das Bankswitching, nach jeder Routine, die auf Bank 1 zugreift, prinzipiell wieder auf die Standardbank 0 zurückzuschalten, hat den Vorteil eines immer exakt definierten Ausgangszustands für weitere Routinen. Es ist dadurch unnötig, sich bei jedem Zugriff auf Bank 0 oder Bank 1 zu überlegen, welche Bank momentan gerade eingeschaltet ist, da beim Eintritt in eine neue Teilroutine des Gesamtprogramms immer Bank 0 eingeschaltet ist. Von dieser »sturen« Lösung abzugehen und damit zusätzliche Fehlerquellen in Kauf zu nehmen, empfehle ich Ihnen jedoch bei besonders zeitkritischen Programmen wie zum Beispiel Sortierroutinen.

Nach dieser kleinen Abschweifung zurück zur Zeichenüberprüfung: wurde auch das letzte Zeichen des Zeichenstrings mit dem eingegebenen Zeichen verglichen, ohne daß eine Übereinstimmung festgestellt wurde, wird zur Eingabeschleife zurückgesprungen, da die gedrückte Taste unzulässig war.

Ist das eingegebene Zeichen jedoch im Zeichenstring enthalten, erfolgt ein Sprung zur Zeichenausgabe:

```
**ZEICHENAUSGABE**
AUSGABE JSR BSOUT
        JMP GET
```

BSOUT gibt ein im Akku enthaltenes Zeichen aus. Das auszugebende Zeichen ist immer noch unverändert im Akku enthalten. Nach der Ausgabe erfolgt die Rückkehr zur Eingabeschleife.

Übernahme der eingegebenen Zeichen:

Nun folgen die für Sie wohl interessantesten Programmteile. Bisher wurden nur bereits erläuterte Routinen des Basic-Interpreters benutzt. Neu war nur die praktische Anwendung des Umschaltens zwischen Bank 0 und Bank 1. Im folgenden lernen Sie jedoch, wie ein String auf dem C 128 angelegt wird und welche Routinen und Zeropageadressen hierfür benötigt werden. Die Übernahme der eingegebenen Zeichen erfolgt in mehreren Schritten:

1. Cursor auf Startposition der Eingabe setzen und einen Pointer auf diese Startposition errechnen
2. Die echte Eingabelänge ermitteln, das heißt alle nachfolgenden Spaces abschneiden, als echte Länge jedoch nur einen Wert akzeptieren, der maximal so groß ist wie die als Parameter übergebene maximale Eingabelänge
3. CHKKOM und GETPOS aufrufen, das heißt einen Zeiger auf die Stringdescriptoren jenes Strings holen, in dem die Eingabe an das Basic übergeben werden soll
4. Speicherplatz für den anzulegenden String reservieren beziehungsweise prüfen, ob ausreichend Platz vorhanden ist
5. Den Bildschirm als logische Datei eröffnen und die Eingabe auf diese Datei legen
6. Die Eingabe vom Bildschirm lesen und in den Übergabestring übertragen
7. Datei schließen und die Eingabe wieder auf Tastatur legen
8. Die Descriptoren für den angelegten String und das Ende des Stringstacks aktualisieren

Cursor auf Startposition setzen/Pointer errechnen

Spalte und Zeile des Eingabebeginns wurden am Programmanfang auf den Stack gerettet. Diese Werte werden nun zurückgeholt und der Cursor auf die dadurch definierte Position gesetzt:

```
*CURSOR AUF STARTPOSITION*
RETURN PLA              ;GERETTETE
TAY                     ;STARTPOSITION
PLA                     ;HOLEN
TAX
CLC                     ;UND CURSOR
JSR SETCRS              ;DARAUF SETZEN
```

Ein Pointer auf diese Startposition wird nun errechnet, indem, wie bereits erwähnt, die momentane Cursorspalte zum Zeiger auf den Beginn der momentanen Cursorzeile addiert wird:

```
*POINTER AUF EINGABESTARTPOS.*
LDA ZEILPOI+1           ;POINTER AUF
STA STARTPOS+1          ;EINGABEBEGINN
LDA ZEILPOI             ;ERZEUGEN
CLC
ADC SPALTE
STA STARTPOS
BCC STARTP
INC STARTPOS+1
```

Nach dieser Addition befindet sich in STARTPOS, STARTPOS+1 ein Pointer, der exakt auf den Beginn der Eingabe zeigt.

Ermitteln der echten Eingabelänge

Um zu demonstrieren, was ich unter »echter« Eingabelänge verstehe, sehen Sie sich folgende möglichen Eingaben an (S = Space). Gehen Sie davon aus, daß das Basic-Programm als maximale Länge einer Eingabe zehn Zeichen vorgab:

C128SSSSSS.....

Commodore128SSSSSS.....

Im ersten Fall wurden nur vier Zeichen eingegeben, die von unserem Programm alle übernommen werden sollen. Die bis zur maximalen Länge folgenden Spaces auf dem Bildschirm sollen jedoch nicht mit in den anzulegenden String übernommen werden (ebensowenig wie beim Basic-Befehl INPUT). Als echte Eingabelänge muß daher eine Vier ermittelt werden. Im zweiten Beispiel wurden zwölf Zeichen eingegeben. Da die maximale Eingabelänge jedoch zehn beträgt und weitere Zeichen nicht übernommen werden sollen, muß unsere Routine als echte Länge eine Zehn ermitteln.

```
*ECHTE EINGABELAENGE ERMITTELN*
STARTP LDY LAENGE
       DEY
LECHT  LDA (STARTPOS),Y
       CMP #32
       BNE OKAY         ;NACHFOLGENDE
       DEY              ;SPACES
       CPY #255         ;ABSCHNEIDEN
       BNE LECHT
```

Die Schleife beginnt mit einem T-Wert von Länge-1 (DEY), das heißt im Beispiel ist Y=9. Zuerst wird daher auf das neunte Zeichen hinter der Startposition zugegriffen, also auf das zehnte Zeichen der Eingabe. Nun wird dieses Zeichen mit SPACE (=Code 32) verglichen. Ist es ungleich SPACE, wird das vorangehende Zeichen mit SPACE verglichen und so weiter.

Bei dem ersten »Nicht-SPACE« wird der Vergleich abgebrochen und zu OKAY gesprungen. Im Y-Register ist die echte Eingabelänge minus eins enthalten.

Stringdescriptoren des Übernahmestring holen

Da das Y-Register die Länge minus eins enthält, erfolgt zuerst eine Korrektur (INY), bevor die echte Länge in LAENGE gespeichert wird. Anschließend werden CHKKOM und GETPOS aufgerufen, wir erhalten also in den Speicherzellen $49, $4A einen Zeiger auf die Descriptoren des Strings, in dem die Eingabe an das Basic übergeben werden soll.

```
*UEBERNAHMEVORBEREITUNGEN*
OKAY INY
     STY LAENGE    ;LECHT RETTEN
     JSR CHKKOM    ;=) ZEIGER AUF
     JSR GETPOS    ;UEBERN.STRING
```

Platz für den anzulegenden String reservieren (STRRS = $9299)

```
     LDA LAENGE
     JSR STRRES    ;PLATZ RESERV.
```

Nun wird der Akku mit der Länge der Eingabe geladen und die Routine STRRES ($9299) aufgerufen. Diese Routine muß vor dem Anlegen eines Strings unbedingt aufgerufen werden. Sie erfüllt mehrere Aufgaben:

1. Sie prüft, ob ausreichend Speicherplatz für einen anzulegenden String mit der im Akku übergebenen Länge vorhanden ist. Wenn nicht, wird eine Garbage Collection durchgeführt. Ist auch anschließend nicht ausreichend Platz vorhanden, wird ein OUT OF MEMORY ERROR ausgegeben.

2. STREND ($35) stellt einen Zeiger auf die Untergrenze des Stringstacks dar. Ein neuer String wird bis zu dieser momentanen Untergrenze angelegt, die nun um die Länge des anzulegenden Strings verringert werden muß. STRRES erledigt auch dieses Herabsetzen des Zeigers STREND um die angegebene Stringlänge. Der neue Wert von STREND ist unser Zeiger auf den Beginn unseres anzulegenden Strings, ab dem dieser Zeichen für Zeichen anlegt.

Bildschirm als logische Datei öffnen/Eingabe auf logische Datei legen

Es erscheint ziemlich umständlich, zum Einlesen der eingegebenen Zeichen den Bildschirm als logische Datei zu definieren und die Eingabe auf diese Datei zu legen. Würde jedoch die beim C 64 von den meisten INPUT-Routinen her verwendete Methode benutzt, die Zeichen ohne Verwendung von Betriebssystem-Routinen direkt vom Bildschirm zu lesen, wäre der Aufwand beim C 128 noch um ein Vielfaches höher: die vom Bildschirm gelesenen Zeichen müssen vor dem Abspeichern im Stringstack in das ASCII-Format konvertiert werden.

Beim C 64 genügt hierzu eine sehr einfache Wandlungsroutine. Der C 128 besitzt jedoch mehrere Zeichensätze – denken Sie an die Umlaute, Akzente, mathematischen Sonderzeichen etc. –, durch die die Wandlung ganz außerordentlich kompliziert und umständlich wird. Es ist daher auf dem C 128 einfacher, wenn möglich die Betriebssystem-Routinen zu verwenden, die diese Wandlung automatisch vornehmen.

```
*BILDSCHIRM: LOG.DATEI EROEFF.*
     LDA #3
     TAX
     TAY
     JSR SETFLS
     LDA #0
     JSR SETNAM
     JSR OPEN
     LDX #3
     JSR CHKIN
```

Der Bildschirm wird als Datei mit der logischen Filenummer drei und der Sekundäradresse definiert, die Länge des Filenamens mit Null angegeben, die Datei geöffnet und die Eingabe auf diese Datei gelegt. Jeder Aufruf der zum Lesen verwendeten Routine BASIN wird sich daher auf den Bildschirm beziehen.

Eingabe lesen/String anlegen

BASIN liest die Zeichen ab der momentanen Cursorposition vom Bildschirm und übergibt sie im Akku. In einer Schleife wird BASIN entsprechend der echten Eingabelänge aufgerufen und das gelesene Zeichen ab der aktualisierten neuen Untergrenze des Stringstacks (STREND) gespeichert. Vor dem Speichern eines gelesenen Zeichens wird auf Bank 1 und anschließend auf Bank 0 geschaltet. Beachten Sie bitte, daß das einmalige und daher zweifellos schnellere Umschalten auf Bank 1 vor der Schleife nicht genügt, da BASIN auf den Bildschirm zugreift, der in Bank 0 liegt und daher auf diese Bank umschaltet.

```
*LESEN/STRING ANLEGEN*
     LDY #0
LIES JSR BASIN
     JSR BANK1
     STA (STREND),Y
     JSR BANK0
     INY
     CPY LAENGE
     BNE LIES
```

Datei schließen/Standardeingabe setzen

Durch CLRCH wird die Eingabe wieder auf das Standardgerät, die Tastatur, gelegt und die geöffnete Datei anschließend mit CLOSE geschlossen.

```
*DATEI SCHLIESSEN/STAND.EING.*
     JSR CLRCH
     LDA #3
     JSR CLOSE
```

Descriptoren aktualisieren

Angenommen, als Übergabestring wurde vom Basic-Programm A$ angegeben. Die Stringdescriptoren von A$ weisen unverändert auf dessen alte Position und beinhalten dessen alte Länge. Wir müssen daher diese Descriptoren noch aktualisieren, also den Längendescriptor mit der Länge und den Positionszeiger mit der Position unseres Strings versehen. Die Länge ist in LAENGE enthalten und die Position entspricht der geänderten Untergrenze STREND des Stringstack, ab der wir unseren String anlegten.

```
*DESCRIPTOREN AKTUALISIEREN*
       JSR BANK1
DESAKT LDY #0
       LDA LAENGE
       STA (ZEIGER),Y
DAKT   LDA STREND,Y
       INY
       STA (ZEIGER),Y
       CPY #2
       BNE DAKT
       JSR BANK0
       RTS     ;ZURUECK INS BASIC
```

Die Eingabe wurde nun komplett übernommen und das Programm kehrt ins Basic zurück. Zur Erinnerung noch einmal das Format des Aufrufs:

```
SYS DEC("0D00"),Länge,Zeile,
Spalte,0,Zeichen$,Rückgabe$
```

Wie Sie wissen, kann GETPOS beliebige Variablentypen verarbeiten. Sie können daher als Zeichensatz- und als Rückgabestring auch Strings aus einem Stringarray angeben.

Die Beschreibung dieser INPUT-Routine ist nun beendet. Sie können auf Ihrem C 128 nun sowohl beliebige Parameter aus dem Basic-Text holen, als auch beliebige Variablen von Maschinensprache aus anlegen. Als Test zum Umgang mit den Basic-Interpreter-Routinen könnten Sie nun zum Beispiel das vorgestellte Programm komfortabler gestalten.

Eine vernünftige INPUT-Routine läßt zum Beispiel – im Gegensatz zu dieser Routine, die zeigen soll, wie mit den entsprechenden Interpreter-Routinen umzugehen ist – keine Cursorbewegungen zu, die aus der vorgegebenen Eingabezone herausführen. Wenn die maximale Eingabelänge zum Beispiel zehn Zeichen beträgt, dürften nur Cursorbewegungen zwischen der Startposition und den nächsten neun Spalten zugelassen werden.

Zudem sollten INSERT und DELETE nur die Zeichen in der Eingabezone verschieben, ohne die restlichen Zeichen in der Zeile zu



beeinflussen und ein Endezeichenstring von Basic übergeben. Nicht nur, wenn RETURN, sondern eine beliebige in diesem String enthaltene Taste gedrückt wird, soll die Eingabe übernommen werden. Wenn das betreffende Zeichen, zum Beispiel CURSOR UP oder CURSOR DOWN dem Basic-Programm in einem String oder einer Integervariablen (ASCII-Code des Zeichens) übergeben wird, lassen sich sehr komfortable Eingabemasken gestalten.

INPUT-Routine

```
a 00d00 85 a5     sta $a5
a 00d02 8a        txa
a 00d03 48        pha
a 00d04 98        tya
a 00d05 48        pha
a 00d06 18        clc
a 00d07 20 f0 ff  jsr $fff0
a 00d0a 20 5c 79  jsr $795c
a 00d0d 20 af 7a  jsr $7aaf
a 00d10 20 20 0b  jsr $0b20
a 00d13 a0 02     ldy #$02
a 00d15 b1 49     lda ($49),y
a 00d17 99 fb 00  sta $00fb,y
a 00d1a 88        dey
a 00d1b 10 f8     bpl $0d15
a 00d1d 20 16 0b  jsr $0b16
a 00d20 20 bf 0d  jsr $0dbf
a 00d23 20 e4 ff  jsr $ffe4
a 00d26 f0 fb     beq $0d23
a 00d28 20 bf 0d  jsr $0dbf
a 00d2b c9 0d     cmp #$0d
a 00d2d f0 18     beq $0d47
a 00d2f a4 fb     ldy $fb
a 00d31 88        dey
a 00d32 20 20 0b  jsr $0b20
a 00d35 d1 fc     cmp ($fc),y
a 00d37 20 16 0b  jsr $0b16
a 00d3a f0 05     beq $0d41
a 00d3c 88        dey
a 00d3d 10 f3     bpl $0d32
a 00d3f 30 df     bmi $0d20
a 00d41 20 d2 ff  jsr $ffd2
a 00d44 4c 20 0d  jmp $0d20
a 00d47 68        pla
a 00d48 a8        tay
a 00d49 68        pla
a 00d4a aa        tax
a 00d4b 18        clc
a 00d4c 20 f0 ff  jsr $fff0
a 00d4f a5 e1     lda $e1
a 00d51 85 a4     sta $a4
a 00d53 a5 e0     lda $e0
a 00d55 18        clc
a 00d56 65 ec     adc $ec
a 00d58 85 a3     sta $a3
a 00d5a 90 02     bcc $0d5e
a 00d5c e6 a4     inc $a4
a 00d5e a4 a5     ldy $a5
a 00d60 88        dey
a 00d61 b1 a3     lda ($a3),y
a 00d63 c9 20     cmp #$20
a 00d65 d0 05     bne $0d6c
a 00d67 88        dey
a 00d68 c0 ff     cpy #$ff
a 00d6a d0 f5     bne $0d61
a 00d6c c8        iny
a 00d6d 84 a5     sty $a5
a 00d6f 20 5c 79  jsr $795c
a 00d72 20 af 7a  jsr $7aaf
a 00d75 a5 a5     lda $a5
a 00d77 20 99 92  jsr $9299
a 00d7a a9 03     lda #$03
a 00d7c aa        tax
a 00d7d a8        tay
a 00d7e 20 ba ff  jsr $ffba
a 00d81 a9 00     lda #$00
a 00d83 20 bd ff  jsr $ffbd
a 00d86 20 c0 ff  jsr $ffc0
a 00d89 a2 03     ldx #$03
a 00d8b 20 c6 ff  jsr $ffc6
a 00d8e a0 00     ldy #$00
a 00d90 20 cf ff  jsr $ffcf
a 00d93 20 20 0b  jsr $0b20
a 00d96 91 35     sta ($35),y
a 00d98 20 16 0b  jsr $0b16
a 00d9b c8        iny
a 00d9c c4 a5     cpy $a5
a 00d9e d0 f0     bne $0d90
a 00da0 20 cc ff  jsr $ffcc
a 00da3 a9 03     lda #$03
a 00da5 20 c3 ff  jsr $ffc3
a 00da8 20 20 0b  jsr $0b20
a 00dab a0 00     ldy #$00
a 00dad a5 a5     lda $a5
a 00daf 91 49     sta ($49),y
a 00db1 b9 35 00  lda $0035,y
a 00db4 c8        iny
a 00db5 91 49     sta ($49),y
a 00db7 c0 02     cpy #$02
a 00db9 d0 f6     bne $0db1
a 00dbb 20 16 0b  jsr $0b16
a 00dbe 60        rts
a 00dbf 48        pha
a 00dc0 a4 ec     ldy $ec
a 00dc2 b1 e0     lda ($e0),y
a 00dc4 49 80     eor #$80
a 00dc6 91 e0     sta ($e0),y
a 00dc8 68        pla
a 00dc9 60        rts
```

Zum Testen dieser Routine geben Sie bitte ein:

Z$="1234567890"+CHR$(29)+CHR$(157)+CHR$(20)+CHR$(148)
SYS DEC("0D00"),10,20,5,0,Z$,A$

Der Cursor wird auf Spalte fünf der Zeile 20 gesetzt; Sie können beliebige Ziffern eingeben und die Eingabe mit CURSOR-RIGHT, CURSOR-LEFT, DEL und INST editieren. Nach RETURN werden maximal zehn Ziffern in die Variable A$ übernommen.

Außer der Variablen- und Stringbehandlung sind die Rechenroutinen des Basic-Interpreters oftmals interessant. So ist es für mich persönlich außerordentlich unangenehm, in Maschinensprache zu multiplizieren und zu dividieren. Da die entsprechenden Interpreter-Routinen nicht sehr schnell arbeiten, ist es für zeitkritische Programmteile notwendig, sich seine eigenen, möglichst optimal an das jeweilige Problem angepaßten Routinen zu schreiben.

In weniger zeitkritischen Programmteilen bietet es sich jedoch an, die bereits vorhandenen Routinen zu nutzen. Die Rechenroutinen des Interpreters arbeiten üblicherweise im Fließkommaformat. In reinen Assemblerprogrammen genügen oftmals Berechnungen, die ausschließlich mit Integerwerten durchgeführt werden. Auch in diesen Fällen können die Interpreter-Routinen genutzt werden, da wir im folgenden Routinen kennenlernen, die Integerwerte ins Fließkommaformat wandeln und weitere, mit denen nach beendeter Berechnung das Ergebnis wieder in einen Integerwert gewandelt wird.

Außer den reinen Rechenroutinen ist für viele Anwendungen eine weitere Routine sehr hilfreich, die einen Fließkommawert auf dem Bildschirm ausgibt. Überlegen Sie sich bitte, welchen Aufwand es erfordert, zum Beispiel die Spalte (Ein-Byte-Integer) und die Zeile (Zwei-Byte-Integer) des Cursors in einer Textverarbeitung ständig »per Hand« in einer Informationszeile auszugeben. Eine weitere Anwendung ist die ständige Ausgabe der momentan erreichten Punktzahl in einem Videospiel.

Da ich beide Anwendungsbeispiele selbst programmiert habe, bevor ich die entsprechenden Interpreter-Routinen kannte, weiß ich deren Vorhandensein nun umso mehr zu schätzen, und auch Ihnen könnte eines Tages ein entsprechendes Problem begegnen.

Alle Rechenroutinen laufen in den sogenannten »Fließkommaakkumulatoren« ab (FAC #1 und FAC #2). FAC #1 befindet sich beim C128 an Adresse $63 bis $69 und FAC #2 an Adresse $6A bis $71. Oftmals wird FAC #1 als FAC und FAC #2 als ARG bezeichnet, da letzterer bei Rechenoperationen mit zwei Fließkommawerten mit dem Argument versehen wird. Im folgenden werde ich diese Bezeichnungen übernehmen.

Auf die Fließkommadarstellung von Zahlen möchte ich in diesem Artikel nicht weiter eingehen, da deren exakte Erläuterung für die Anwendung der Rechenroutinen nicht unbedingt nötig ist und ich selbst zugegebenermaßen mit der

Fließkommaarithmetik nicht sonderlich vertraut bin.

Wichtig für uns ist, daß jeder Fließkomma-Akkumulator gleich aufgebaut ist. Für unsere Anwendungsbeispiele benötigen wir folgende Adressen:

```
$63     = FAC, Exponent
$64-$67 = FAC, Mantisse
$68     = FAC, Vorzeichen

$6A     = ARG, Exponent
$6B-$6E = ARG, Mantisse
$6F     = ARG, Vorzeichen
```

In den folgenden Beispielen werde ich drei Fließkommazahlen verwenden, mit denen die verschiedenen Programmbeispiele arbeiten werden:

dez.	hex	Fließkommaformat
8	$08	84 80 00 00 00
10	$A0	84 A0 00 00 00
4098	$1002	8D 80 00 00 00

Zur Erläuterung der verschiedenen Rechenroutinen würde es genügen, FAC und ARG mit den jeweiligen Werten zu laden, indem der eingebaute Monitor benutzt wird, die Rechenroutine aufzurufen und das Ergebnis wiederum mit dem Monitor zu betrachten. Nachdem ich diese Methode anwandte und völlig unsinnige Ergebnisse erhielt, mußte ich feststellen, daß der Monitor selbst FAC verwendet (für seinen Zeiger auf die momentane Adresse, zum Beispiel beim Assemblieren/Disassemblieren).

Im folgenden werden wir daher zwangsläufig unsere Beispielwerte mit einem Programm in die Fließkommaakkumulatoren schreiben und auch das Ergebnis vom Programm retten lassen, bevor wir den Monitor wieder benutzen.

Führen Sie bitte auch diesmal wieder die notwendige Initialisierung durch, wenn Sie nach den vorigen Beispielen den Rechner ausgeschaltet oder einen Reset durchgeführt haben. Als Initialisierungs-Programm genügt die Routine ohne die Unterprogramme zum Umschalten der Speicherbänke, da wir nicht auf Bank 1 zugreifen müssen.

FAC-Inhalt retten

Die folgende Maschinenroutine rettet den Inhalt des Fließkommaakkus Nummer 1, der beim Aufruf des Monitors durch diesen verändert wird:

```
a 00b30  a2 04     ldx #$04
a 00b32  b5 63     lda $63,x
a 00b34  9d 00 0d  sta $0d00,x
a 00b37  ca        dex
a 00b38  10 f8     bpl $0b32
a 00b3a  60        rts
```

Fließkomma nach Integer wandeln ($8CC7)

Diese Routine wandelt einen Fließkommawert, der sich im FAC befindet, in eine Zwei-Byte-Integerzahl, die wiederum im FAC abgelegt wird (in $66/$67 = High/Low).

```
a 00b40  a2 04     ldx #$04
a 00b42  bd 50 0b  lda $0b50,x
a 00b45  95 63     sta $63,x
a 00b47  ca        dex
a 00b48  10 f8     bpl $0b42
a 00b4a  20 c7 8c  jsr $8cc7
a 00b4d  4c 30 0b  jmp $0b30
a 00b50  84 a0     sty $a0
a 00b52  00        brk
a 00b53  00        brk
a 00b54  00        brk
```

Da das Ende dieses Listings ein wenig ungewöhnlich erscheinen mag, will ich die letzten Befehle genauer erläutern: Um eine Integerzahl in eine Fließkommazahl oder eine Fließkommazahl in eine Integerzahl zu wandeln, muß erstere im FAC #1 abgelegt sein. In diesem Beispiel verwende ich die Dezimalzahl zehn, die sich, im Fließkommaformat abgelegt, am Programmende befindet:

```
m 0b50 84 a0 00 00 00
```

Das Programm überträgt diese Werte nach FAC #1. Nach dem Aufruf der Routine $8CC7 befindet sich die entsprechende Integerzahl in den Adressen $66, $67 (High/Low) und wird durch das Unterprogramm $0B30 nach $0D00 bis $0D04 übertragen und vor dem Überschreiben durch den Monitor gerettet.

Rufen Sie die Routine mit »SYS DEC("0B40")« auf und schauen Sie sich $0D00... mit dem Monitor an:

```
m 0d00 84 00 00 00 A0
```

In $0D03 und $0D04 befindet sich die Integerzahl $00A0.

2-Byte-Integer (vorzeichenbehaftet) nach Fließkomma wandeln ($8C70)

Es gibt zwei Arten der Darstellung von Integerzahlen: vorzeichenbehaftet und positiv. Als Assemblerprogrammierer benötigen Sie üblicherweise nur die positive Darstellung, bei der alle 16 Bits für den Absolutwert einer Zahl verwendet werden, die damit im Bereich zwischen $0000 bis $FFFF liegen kann.

Bei der vorzeichenbehafteten Darstellung wird Bit-16 als Vorzeichen betrachtet (gesetzt = negativ). Eine Vorzeichenbehaftete Zwei-Byte-Integerzahl besitzt daher einen Wert zwischen -$7FFF und +$7FFF. Diese Routine wird zum Beispiel beim C 64 zur Ausgabe von FRE(0) verwendet, in diesem Fall leider fälschlicherweise, da jeder C 64-Programmierer das Problem kennt, daß bei Werten, die größer sind als $7FFF der freie Speicherplatz angeblich negativ ist.

Die Integerzahl wird dieser Routine in den Speicherzellen $64, $65 (High/Low) übergeben. Das X-Register muß vor dem Aufruf unmittelbar mit $90 geladen werden.

```
a 00b60  a9 01     lda #$01
a 00b62  a2 00     ldx #$00
a 00b64  85 64     sta $64
a 00b66  86 65     stx $65
a 00b68  a2 90     ldx #$90
a 00b6a  20 70 8c  jsr $8c70
a 00b6d  4c 30 0b  jmp $0b30
```

Nach dem Aufruf mit »SYS DEC ("0B60")« befindet sich die übergebene Zahl +$0100 im Fließkommaformat im FAC #1. Die Routine $0B30 überträgt sie nach $0D00 bis $0D04, wovon Sie sich wiederum mit dem Monitor überzeugen können.

2-Byte-Integer (positiv) nach Fließkommaformat wandeln ($8C75)

Wahrscheinlich werden Sie diese Routine, die die üblichen positiven Integerwerte ins Fließkommaformat wandelt, häufiger in Ihren Programmen verwenden. Auch dieser Routine wird die zu wandelnde Integerzahl in den Speicherzellen $64, $65 übergeben und auch das X-Register vor dem Aufruf unmittelbar mit $90 geladen. Weiterhin muß das Carry-Flag vor dem Einsprung gesetzt werden.

```
a 00b80  a9 ff     lda #$ff
a 00b82  a2 01     ldx #$01
a 00b84  85 64     sta $64
a 00b86  86 65     stx $65
a 00b88  a2 90     ldx #$90
a 00b8a  38        sec
a 00b8b  20 75 8c  jsr $8c75
a 00b8e  4c 30 0b  jmp $0b30
```

Dieses Demoprogramm übergibt den Wert $FF01. Nach dem Aufruf mit »SYS DEC("0B80")« befindet sich die gerettete Fließkommazahl in $0D00 bis $0D04:

```
m 0d00  90 ff 01 00 00
```

Wenn Sie in Ihren Programmen nur mit Integerzahlen rechnen, werden Sie sich fragen, was Sie mit Routinen zur Umwandlung von Integer- in Fließkommazahlen anfangen sollen. Zum Beispiel, die Integerzahl auf dem Bildschirm ausgeben. Eine solche Ausgabe ist unter Verwendung von zwei weiteren Routinen möglich, jedoch erst, nachdem die Integerzahl ins Fließkommaformat gewandelt wurde:

FAC #1 nach ASCII wandeln ($8E42)

Diese Routine legt eine Fließkom-



mazahl, die sich im FAC #1 befindet, als String (ASCII-Format) ab, der sich wiederum im FAC #1 befindet. Interessant wird diese Wandlung in Verbindung mit einer weiteren Routine:

String in FAC #1 ausgeben ($55E2)

$55E2 gibt einen String, der sich im FAC #1 befindet, auf dem Bildschirm ab der momentanen Cursorposition aus.

Beide Routinen benötigen zur Vorbereitung nur die Fließkommazahl in FAC #1 ($8E42) beziehungsweise den String in FAC #1 ($55E2) und können ohne Übergabe weiterer Parameter aufgerufen werden. Wir besitzen nun alle erforderlichen Kenntnisse, um eine beliebige Integerzahl auf dem Bildschirm auszugeben, wie es für jede Textverarbeitung erforderlich ist.

Zuerst übergeben wir eine positive Integerzahl an die Speicherzellen $64, $65 und rufen $8C75 auf, die die Wandlung ins Fließkommaformat vornimmt. Anschließend wird $8E42 aufgerufen und die Zahl dadurch in einen String umgewandelt, der mit $55E2 ab der momentanen Cursorposition ausgegeben wird:

```
a 00ba0  a9 ff     lda #$ff
a 00ba2  a2 01     ldx #$01
a 00ba4  85 64     sta $64
a 00ba6  86 65     stx $65
a 00ba8  a2 90     ldx #$90
a 00baa  38        sec
a 00bab  20 75 8c  jsr $8c75
a 00bae  20 42 8e  jsr $8e42
a 00bb1  4c e2 55  jmp $55e2
```

Das Demoprogramm verwendet die Zahl $FF01, also dezimal 65281. Wenn Sie es mit »SYS DEC("0BA0")« starten, wird eben diese Zahl auf dem Bildschirm ausgegeben. Äußerst angenehm an diesen Routinen ist, daß Sie zwar nicht ohne das Fließkommaformat auskommen, der Programmierer selbst mit diesem jedoch nicht im geringsten in Berührung kommt, sondern nur eine Integerzahl übergibt.

2-Byte-Integer (positiv) ausgeben ($8E32)

Die Wandlungs- und Ausgaberoutinen, die zur Ausgabe einer Integerzahl nacheinander aufgerufen werden, können selbstverständlich unabhängig voneinander auch für andere Zwecke eingesetzt werden. Zur Ausgabe einer Integerzahl auf dem Bildschirm gibt es eine weitere, besser geeignete Routine, die ab der Adresse $8E32 liegt. Dieser Routine wird die betreffende Integerzahl im Akku und im X-Register übergeben (Akku = Low-, X = High-Byte). Sie legt diese Werte in den Speicherzellen $64, $65 ab und ruft die beschriebenen Routinen in der zur Bildschirmausgabe notwendigen Reihenfolge auf. Die Ausgabevorbereitungen beschränken sich daher auf das Laden dieser Register. Danach kann die Routine aufgerufen werden:

```
a 00bc0  a9 ff     lda #$ff
a 00bc2  a2 01     ldx #$01
a 00bc4  4c 00 00  jmp $8e32
```

Diese drei Programmzeilen besitzen die gleiche Funktion wie das vorige Demoprogramm. Sie geben ebenfalls den Wert $FF01, also die Dezimalzahl 65281 auf dem Bildschirm aus. Der Aufruf erfolgt mit »SYS DEC("0BC0")«.

Alle im folgenden beschriebenen Routinen führen Berechnungen mit zwei Fließkommazahlen durch (Addition, Division etc.), die sich in FAC #1 und FAC #2 befinden und speichern das Ergebnis im FAC #1. Da uns nun bekannt ist, auf welche Weise Integerzahlen ins Fließkommaformat gewandelt werden, können wir sie ebenso für Berechnungen mit zwei Integerzahlen verwenden, indem wir diese mit den entsprechenden Routinen ins Fließkommaformat wandeln.

Diese Anwendung kann vor allem in Maschinenprogrammen, in denen häufig multipliziert und dividiert werden muß, das Schreiben eigener Routinen ersparen. Bei zeitkritischen Programmteilen sollte die Verwendung der Fließkommaarithmetik jedoch unbedingt vermieden werden, da die Integerzahlen zuerst aufwendig gewandelt werden müssen und anschließend die noch weit aufwendigeren Fließkommarechenoperationen erfolgen. In solchen Fällen ist es angebrachter, eigene Integer-Rechenroutinen zu schreiben.

Gerade unsinnig wird die Verwendung der vorgestellten Routinen in speziellen Fällen wie zum Beispiel der Multiplikation oder Division einer (Zwei-Byte-) Integerzahl mit zwei, vier etc., da diese mit wenigen Rotations- und Verschiebebefehlen erledigt werden kann. Merken Sie sich daher bitte: für Rechenoperationen mit den in Assemblerprogrammen zumeist gebrauchten Integerzahlen sollte die eingebaute Fließkommaarithmetik nur in zeitunkritischen Fällen verwendet werden!

FAC = FAC + ARG ($8848)

Diese Routine addiert zwei in FAC #1 (=FAC) und FAC #2 (=ARG) hinterlegte Fließkommazahlen.

```
a 00bd0  a2 04     ldx #$04
a 00bd2  bd e8 0b  lda $0be8,x
a 00bd5  95 63     sta $63,x
a 00bd7  bd ed 0b  lda $0bed,x
a 00bda  95 6a     sta $6a,x
a 00bdc  ca        dex
a 00bdd  10 f3     bpl $0bd2
a 00bdf  20 48 88  jsr $8848
;FAC = FAC + ARG
a 00be2  20 42 8e  jsr $8e42
a 00be5  4c e2 55  jmp $55e2
a 00be8  84 80     sty $80
a 00bea  00        brk
a 00beb  00        brk
a 00bec  00        brk
a 00bed  8d 80 10  sta $1080
a 00bf0  00        brk
a 00bf1  00        brk
```

Rufen Sie die Routine mit »SYS DEC("0BD0")« auf. Die Fließkommazahlen ($08=8 und $1002=4098) befinden sich am Ende des Programms und werden in einer Schleife an den FAC und ARG übergeben. Anschließend wird die Additions-Routine, die ASCII-Wandlungs- und die Ausgabe-Routine aufgerufen. Wir erhalten die Ausgabe der Zahl 4106 = 8 + 4098.

FAC = ARG – FAC ($8831)

Auch diese Routine benötigt zwei in FAC und ARG hinterlegte Fließkommazahlen. Achten Sie bitte darauf, daß FAC von ARG subtrahiert wird und nicht umgekehrt. Um die Anwendung zu demonstrieren, ersetzen Sie bitte einfach im letzten Programm den Befehl »jsr 8848« durch »jsr 8831«:

```
...
...
...
a 00bdc  ca        dex
a 00bdd  10 f3     bpl $0bd2
a 00bdf  20 48 88  jsr $8848
;FAC = FAC - ARG
a 00be2  20 42 8e  jsr $8e42
a 00be5  4c e2 55  jmp $55e2
...
...
...
```

Als Ergebnis erhalten wir 4090 = 4098 – 8.

FAC = ARG * FAC ($8A27)

Der Aufruf unterscheidet sich in keiner Weise von den vorangehenden Arithmetik-Routinen. Ersetzen Sie daher »jsr 8848« durch »jsr 8a27«:

```
...
...
...
a 00bdc  ca        dex
a 00bdd  10 f3     bpl $0bd2
a 00bdf  20 48 88  jsr $8a27
;FAC = FAC * ARG
a 00be2  20 42 8e  jsr $8e42
a 00be5  4c e2 55  jmp $55e2
...
...
...
```

Als Ausgabe erhalten wir 32784 = 8 * 4098.

FAC = ARG / FAC ($8B4C)

Ändern Sie nun »jsr 8a27« ab in »jsr 8b4c«. Sie erhalten die Ausgabe von 512.25 = 4098 / 8. Achten Sie bitte wie bei der Subtraktion auch hier auf die Reihenfolge von Dividend und Divisor:

```
 ...
 ...
 ...
a 00bdc  ca       dex
a 00bdd  10 f3    bpl $0bd2
a 00bdf  20 48 88 jsr $8b4c
;FAC = ARG / FAC
a 00be2  20 42 8e jsr $8e42
a 00be5  4c e2 55 jmp $55e2
 ...
 ...
 ...
```

Zum Abschluß möchte ich noch zwei Spezialroutinen vorstellen, die in den entsprechenden Fällen zum einen weniger Vorbereitungen benötigen und zum anderen erheblich schneller arbeiten als die Benutzung von FAC=FAC*ARG oder FAC=ARG/FAC.

FAC = FAC * 10 ($8B17)

Sollte in Ihren Programmen eine entsprechende Anwendung gegeben sein, so ersparen Sie sich mit dieser Routine die Wandlung von Zehn ins Fließkommaformat.

```
a 00bd0  a2 04    ldx #$04
a 00bd2  bd e8 0b lda $0be3,x
a 00bd5  95 63    sta $63,x
a 00bd7  ca       dex
a 00bd8  10 f3    bpl $0bd2
a 00bda  20 48 88 jsr $8b17
;FAC = FAC * 10
a 00bdd  20 42 8e jsr $8e42
a 00be0  4c e2 55 jmp $55e2
a 00be3  84 80    sty $80
a 00be5  00       brk
a 00be6  00       brk
a 00be7  00       brk
```

Der Aufruf mit »SYS DEC("0BD0")« führt zur Ausgabe von 80 = 8 * 10.

FAC = FAC / 10 ($8B38)

```
 ...
 ...
 ...
a 00bd7  ca       dex
a 00bd8  10 f3    bpl $0bd2
a 00bda  20 48 88 jsr $8b17
;FAC = FAC * 10
a 00bdd  20 42 8e jsr $8e42
a 00be0  4c e2 55 jmp $55e2
 ...
 ...
 ...
```

Nach dem Aufruf erhalten wir die Ausgabe von 0,8 = 8 / 10.

Wollen Sie sich die Fließkommaarithmetik auch für das Rechnen mit Integerzahlen zunutze machen, sieht der Ablauf prinzipiell wie folgt aus:

1. Akku und X-Register mit dem ersten Wert laden
2. Eine eigene Unterroutine aufrufen, die die Registerinhalte benutzt, um sie der Wandlungsroutine INTEGER nach FLIESSKOMMA zu übergeben und diese aufruft
3. Eine weitere eigene Unter-Routine aufrufen, die die FAC nach ARG kopiert
4. Akku und X-Register mit dem zweiten Wert laden
5. Die Unter-Routine von 2. aufrufen
6. Die entsprechende Arithmetik-Routine aufrufen
7. Ein eigenes Unterprogramm aufrufen, das die Wandlungsroutine FLIESSKOMMA nach INTEGER aufruft und den Inhalt der Speicherzellen $66, $67 dem Hauptprogramm im Akku und im X-Register übergibt

Nachdem Sie diese kleinen Unterprogramme geschrieben haben, genügen zur Multiplikation von $00A5 mit $02B2 folgende Befehle:

```
LDA #$B2
LDX #$02
JSR ROUTINE1      ;$02b2 nach
                  Fließkomma wandeln
JSR ROUTINE2      ;FAC in ARG
                  kopieren
LDA #$A5
LDX #$00
JSR ROUTINE1      ;$00a5 nach
                  Fließkomma
                  wandeln
JSR ARITHMETIKROUTINE
                  ;Operation
                  durchführen
JSR ROUTINE3      ;Fließkomma nach
                  Integer wandeln
                  und nach A,X
```

(S. Baloui / ah)

Die wichtigsten Interpreterroutinen des C128

Routine	Parameter hin	Parameter zurück	C128
Parameter aus dem Basic-Text holen:			
CHKKOM	..,(Komma)	keine	$795C
GETBYT	..,(Byte)	Byte im X-Register	$87F4
GETBYT MIT CHRGET	..,(Byte)	Byte im X-Register	$87F1
FRNUM	..,(num.Ausdruck)	Ergebnis in FAC #1	$77D7
ADRFOR	FAC #1: num. Ausdruck	Adresse in Y/Akku und in $16,$17	$8815
GETADR	..,(Adresse)	Adresse in Y/Akku und in $16,$17	$880F
ADRBYT	..,(Adresse),(Byte)	Byte im X-Register Adresse in $16,$17	$8803
Anlegen von Variablen:			
GETPOS	..,(belieb.Variable)	Zeiger auf die Var. in Akku/Y und $49,$4A Var. Name in $47,$48	$7AAF
STRRES	Stringlänge im Akku	Ev. OUT OF-MEMORY STREND ($35,$36) um Länge verringern	$9299
TYPFLAGS	keine	$0F:$00=num.,$FF=Str. $10:$00=Real,$80=Int$10	$0F
Routinen zur Fließkommaarithmetik:			
FLIESSK.NACH INTEGR	Fließk. Zahl in FAC	Integerz. in $66,$67	$8CC7
INT. MIT VORZ. NACH FLIESSK.	Int.Zahl in $64, $65 X=$90	Fließk. Zahl in FAC	$8C70
INT. NACH FLIESSKOMMA	Int. Zahl in $64, $65 X=90, SEC	Fließk. Zahl in FAC	$8C75
FAC NACH ASCII	Fließkommazahl in FAC	Zahlenstring in FAC	$8E42
AUSGABE VON STRING IM FAC	Zahlenstring in FAC	Ausgabe auf Screen	$55E2
AUSGABE VON POS. INTEGERZAHL	Integerzahl in Akku/X	Ausgabe auf Screen	$8E32
FAC = FAC + ARG	Fließkommaz. in FAC/ARG	Ergebnis in FAC	$8848
FAC = ARG - FAC	Fließkommaz. in FAC/ARG	Ergebnis in FAC	$8831
FAC = ARG * FAC	Fließkommaz. in FAC/ARG	Ergebnis in FAC	$8A27
FAC = ARG / FAC	Fließkommaz. in FAC/ARG	Ergebnis in FAC	$8B4C
FAC = FAC * 10	Fließkommazahl in FAC	Ergebnis in FAC	$8B17
FAC = FAC / 10	Fließkommazahl in FAC	Ergebnis in FAC	$8B38



Tips und Tricks zu C 128

Um jedem C128-Besitzer die Programmierung in Maschinensprache zu erleichtern, schildern wir hier die ersten Erfahrungen mit diesem Computer. Durch diese Erfahrungen werden Sie schneller mit dem C128 zurechtkommen.

Das »Bankswitching«, also das Umschalten zwischen mehreren sich überlagernden Speicherbereichen, ist bereits vom C64 her bekannt, bei dem auf diese Weise RAM-Bereiche unter dem ROM genutzt werden können. Beim C64 war es jedoch nicht unbedingt erforderlich, über Speicherkonfigurationen und deren Änderung Bescheid zu wissen. Viele Assemblerprogrammierer dürften sich – ebenso wie ich selbst – so weit wie möglich um diese etwas »ominöse« Geschichte herumgedrückt haben.

Die Programmierung des C128 in Maschinensprache ist leider ohne Kenntnis des Bankswitching kaum möglich. Wie Sie bereits wissen, verwendet der Basic-Interpreter Bank 0 für das Basic-Programm und Bank 1 für die Variablenspeicherung. Der Interpreter ist daher gezwungen, ständig zwischen diesen 64 KByte RAM-Bänken hin- und herzuschalten.

Das gleiche Problem stellt sich für den Assemblerprogrammierer. Eine beliebte Anwendung von Assemblerprogrammen sind Unterroutinen, die mit dem Basic zusammenarbeiten sollen, zum Beispiel um besonders zeitkritische Programmteile zu beschleunigen, oder aber, um nicht vorhandene Basic-Befehle zu implementieren. Werden solche Routinen auf dem C64 geschrieben, kann Sie der Programmierer problemlos in einen freien, vom Überschreiben durch Basic geschützten Speicherbereich legen (zum Beispiel $C000 bis $CFFF).

Der C128 erfordert auch bei kleineren Assembler-Routinen einen weitaus höheren Aufwand. So stellt bereits die Frage, in welchen Speicherbereich eine Routine gelegt werden soll, größere Probleme. Es stehen zwar 128 KByte zur Verfügung, jedoch kein größerer Speicherbereich, der vom Basic mit Sicherheit verschont bliebe. Bank 0 steht bis $FFFF für den Basic-Text, und Bank 1 ebenfalls bis $FFFF für Variablen zur Verfügung. Weiterhin muß in vielen Fällen eine Assembler-Routine auf den Basic-Text und auch auf die Variablen zugreifen, was ohne Umschaltung der Konfiguration nicht möglich ist. Zum Beispiel muß eine Sortierroutine Parameter aus dem Basic-Text lesen (Name des zu sortierenden Arrays), der eigentliche Sortiervorgang findet anschließend in Bank 1, der Variablenbank, statt.

Die ersten Gehversuche

Da ich zusammen mit einem Programmiererkollegen kürzlich vor der Aufgabe stand, ein Basic-Programm, das verschiedene Assembler-Routinen verwendet, vom C64 auf den C128 umzuschreiben, waren wir gezwungen, uns mit diesen – für uns völlig ungewohnten – Problemen auseinanderzusetzen. In diesem Artikel werde ich beschreiben, welche Lösungswege wir sahen, wieder verwarfen, und wie es uns nach unzähligen Fehlschlägen tatsächlich gelang, alle benötigten Routinen umzuschreiben. Ich bin der Ansicht, daß unsere Vorgehensweise typisch war und dieser Artikel es vielen C128-»Neulingen« erspart, unsere zum Großteil frustrierenden Erfahrungen nachzuvollziehen.

Begonnen hatte alles mit einem C128, dem Handbuch und einem Exemplar des »C128 Intern«. Unser erster Schritt war ein Testprogramm, mit dem das Umschalten zwischen den Speicherbänken erprobt werden sollte. Das Programm sollte eine Speicherstelle in Bank 1 lesen. Wir gaben es mit dem integrierten Monitor in Bank 0 ein, und legten es – wie vom C64 gewohnt – nach $C000. Nach dem Aufruf sollte es durch Beschreiben des Konfigurationsregisters $FF00 auf Bank 1 umschalten, die gewünschte Speicherstelle in den Akku einlesen und mit einem BRK in den Monitor zurückkehren.

Nachdem wir feststellen mußten, daß dieses einfache Testprogramm abstürzte, war klar, daß uns C64-Programmierern gewaltige Umstellungen bevorstanden. Der Grund für den Absturz war schnell gefunden: nach dem Beschreiben von $FF00 und damit dem Umschalten auf Bank 1 folgte der Befehl »LDA $1000«, der den Inhalt dieser Speicherstelle lesen sollte. Da unser Programm jedoch in Bank 0 lag, sägten wir uns mit dem Umschalten auf Bank 1 gewissermaßen »den eigenen Ast ab«, da unser Programm – und damit auch der Ladebefehl – nach dem Umschalten auf Bank 1 für den Prozessor nicht mehr existent war.

Dieser Mißerfolg machte uns immerhin klar, daß wir zu naiv an den neuen Computer herangegangen waren. Wir hatten keine Chance, ohne nähere Kenntnis des C128, unsere Routinen umzuschreiben. Nachdem wir einige Zeit damit verbrachten, das Handbuch und das C128 Intern zu studieren, war klar, daß es prinzipiell zwei Methoden gibt, auf verschiedene Speicherbänke zuzugreifen, ohne daß ein Programm unmittelbar nach dem Umschalten abstürzt:

1. Benutzung der Routinen FETCH, STASH und CMPARE:

Diese Routinen benutzt auch der Basic-Interpreter. Nach dem Einschalten des Computers ist der Bereich $0000 bis $03FF als gemeinsamer Bereich von Bank 0 und Bank 1 definiert. Unabhängig davon, welche Bank eingeschaltet ist, »sieht« der Prozessor immer die gleichen Programme beziehungsweise Daten in diesem Bereich. Dieses Konzept der gemeinsamen Speicherbereiche kann man sich durch die Vorstellung veranschaulichen, daß jede Änderung des Bereichs $0000 bis $03FF in der einen Bank sofort in die andere Bank kopiert wird, die Inhalte daher in jedem Moment identisch sind. Dieses Konzept entspricht zwar nicht dem tatsächlichen physischen Ablauf, ist jedoch eine für die Praxis völlig ausreichende Vorstellung.

Die genannten Routinen werden nach dem Einschalten des Compu-

ters aus dem ROM in diesen gemeinsamen Bereich kopiert. Ihre Benutzung erfordert eine recht langwierige Parameterübergabe. Unter anderem wird die gewünschte Speicherkonfiguration übergeben, auf die zugegriffen werden soll. Nach der Parameterübergabe und dem Aufruf schaltet die jeweilige Routine die benötigte Konfiguration ein, führt in der ausgewählten Bank die gewünschte Operation aus (Lesen, Schreiben, Vergleichen) und schaltet danach auf die alte Speicherkonfiguration zurück.

Der Einsprung in eine dieser Routinen findet in das Kernel statt. Danach wird jedoch in die Kopie, also in den gemeinsamen Speicherbereich verzweigt. Da das Umschalten in diesem gemeinsamen Bereich durchgeführt wird, findet kein Absturz statt, da die Routine für den Prozessor in jeder Bank existent ist. Diese Routinen sind daher das »Verbindungsglied« zwischen Programmen und Daten, die in verschiedenen Bänken liegen.

Der gemeinsame Speicherbereich

Ich erspare es mir, diese Routinen mit allen übergebenen und rückübergebenen Parametern zu beschreiben, da inzwischen mehrere Bücher kompetenterer Programmierer auf dem Markt sind, die das Innenleben des C 128 detailliert beschreiben.

2. Assembler-Routinen in einen gemeinsamen Speicherbereich legen:

Wie erwähnt, wird beim Einschalten des C 128 ein bestimmter Speicherbereich als gemeinsam definiert. Über das Register $D506, das im I/O-Bereich liegt, kann der Programmierer jedoch selbst bestimmen, ob er gemeinsame Bereiche definieren will, wo und wie groß diese sein sollen.

Es existieren jedoch mehrere Begrenzungen, zum Beispiel dadurch, daß ein gemeinsamer Bereich an einem Ende des Speichers beginnen muß (entweder ab $0000 aufwärts oder ab $FFFF abwärts) und durch die maximale Größe eines solchen Bereichs von 32 KByte.

Die zweite Möglichkeit besteht somit darin, einen Speicherbereich als gemeinsam zu deklarieren, der groß genug ist, alle benötigten Assembler-Routinen aufzunehmen.

Da die Routinen FETCH etc. langwierige Parameterübergaben erfordern, ein Programm daher länger wird und zusätzliche Fehlerquellen entstehen, waren mein Kollege und ich uns einig, die zweite Möglichkeit zu verwenden. Die Frage war nun, welcher Speicherbereich als gemeinsam deklariert und für die Routinen verwendet werden sollte. Wir entschieden uns dafür, den oberen Speicherbereich bis $FFFF zu verwenden. Voraussetzung dafür war natürlich, den Zeiger auf das Ende des Basic-Textes herabzusetzen, um die Routinen vor dem Überschreiben zu schützen. Zusätzlich war es erforderlich, die Zeiger auf den Anfang des Stringbereichs ebenfalls herabzusetzen. Der Stringstack beginnt ab $FFFF in Bank 1. Wenn nun zum Beispiel der Speicherbereich $F000 bis $FFFF als gemeinsamer Bereich definiert wird, führt jedes Anlegen eines Strings dazu, daß die Assembler-Routinen in dem entsprechenden Bereich von Bank 0 durch den String überschrieben würden.

Nach den frustrierenden Erfahrungen mit dem erstem Testprogramm versuchten wir nun, möglichst alle Konsequenzen unseres Plans zu überdenken. Dabei stellte sich ein neues Problem: Im gleichen Bereich befindet sich das Kernel, das zum Zugriff auf unsere Routinen ausgeschaltet werden mußte. Diese Routinen benötigten jedoch selbst mehrere Kernel-Routinen.

Zur Benutzung von ROM-Routinen trotz ausgeblendetem ROM stehen die Routinen JRSFAR und JMPFAR zur Verfügung. Bei der Benutzung dieser Routinen muß das Kernel-ROM eingeschaltet sein. Die Routinen selbst befinden sich zwar nicht im Kernel, verzweigen jedoch zu einer Kernel-Routine, was bei ausgeblendetem Kernel natürlich zu einem Absturz führen würde.

JSRFAR stellt daher leider für Programme, die sich unter dem Kernel befinden und dessen Routinen verwenden wollen, keine brauchbare Lösung dar. Denkbar ist jedoch ein Umweg, mit dem das Ziel gewissermaßen über mehrere Ecken angesteuert wird:

Zwei gemeinsame Bereiche werden definiert, einer am unteren und einer am oberen Ende des Speichers. In den unteren Bereich, der nicht vom ROM überlagert wird, wird eine kleine Routine gelegt, die den Aufruf einer vom Hauptprogramm benötigten Kernel-Routine übernimmt. Das Hauptprogramm übergibt alle Parameter (zum Beispiel die Adresse der Kernel-Routine) und ruft die Routine im unteren Bereich auf. Diese blendet das Kernel ein und dadurch zwangsläufig das Hauptprogramm aus. Nun wird JSRFAR aufgerufen. Nach Durchführung der jeweiligen Kernel-Routine wird mit einem RTS zu der Routine im unteren gemeinsamen Bereich zurückgekehrt. Diese blendet nun das Kernel wieder aus und das darunterliegende Hauptprogramm ein; anschließend erfolgt die Rückkehr in dieses Hauptprogramm.

Wie Sie sehen – und auch wir einsehen mußten – ist es ohne größeren Aufwand nicht möglich, ein Assemblerprogramm, das Kernel-Routinen verwendet, unter eben dieses Kernel zu legen. Die Umschaltung zwischen Kernel und darunterliegendem Programm führt zu sehr umständlichen Routinen.

Die einzige von uns gefundene Möglichkeit, kleinere Assembler-Routinen auf dem C 128 zu schreiben, ohne an derartigen Problemen zu scheitern, besteht darin, die Programme in einen Bereich zu legen, der nicht vom ROM - das eventuell benötigt wird - überlagert und als gemeinsamer Bereich von Bank 0 und Bank 1 deklariert wird.

Bankswitching

Ein solcher Bereich, der für die meisten Assembler-Routinen ausreichend Platz bietet und unbenutzt ist, ist der Bereich von $1300 bis $17FF. Um Probleme mit dem Umschalten zwischen den Bänken zu vermeiden, wird ein gemeinsamer Bereich definiert, der von $0000 bis $1FFF reicht. Da der Basic-Text jedoch ab $1C00 beginnt, muß der Zeiger auf den Basic-Anfang auf $2000 verstellt werden, um Überschneidungen zwischen dem Basic-Text in Bank 0 und den Variablen in Bank 1 zu vermeiden.

Zusätzlich muß auch der Zeiger auf den Beginn der Variablentabelle verstellt werden – auch auf $2000 –, da diese ab $0800 beginnt und ebenfalls ab $0800, jedoch in Bank 0, der Bildschirmspeicher liegt. Überschneidungen zwischen Variablentabelle und Bildschirmspeicher führen sonst bei jeder Ausgabe auf dem Bildschirm zu Änderungen der Variablentabelle, also zu ihrer Zerstörung.

Ich möchte Ihnen nun zwei kleine Initialisierungs-Routinen vorstellen, die die gewünschten Aufgaben erledigen, eine Basic- und eine



Assembler-Routine:
1. Beginn des Basic-Textes und der Variablentabelle verschieben:

```
10 poke 46,dec("20"):poke 48,dec
("20")
20 poke dec("2000"),0
30 clr
40 new
```

Wenn Sie dieses Basic-Programm eingeben und starten, beginnt sowohl der Basic-Text als auch die Variablentabelle ab $2000.

2. Definition eines gemeinsamen Speicherbereiches:

```
a 00b00  ad  00 ff  lda $ff00
a 00b03  48         pha
a 00b04  a9  00     lda #$00
a 00b06  8d  00 ff  sta $ff00
a 00b09  ad  06 d5  lda $d506
a 00b0c  09  06     ora #$06
a 00b0e  8d  06 d5  sta $d506
a 00b11  68         pla
a 00b12  8d  00 ff  sta $ff00
a 00b15  60         rts
```

Dieses kleine Maschinenprogramm rettet zuerst die aktuelle Speicherkonfiguration, blendet nun den I/O-Bereich ein und definiert im RAM-Konfigurationsregister den Bereich $0000 bis $1FFF als gemeinsamen Bereich von Bank 0 und Bank 1. Vor dem Rücksprung aus der Routine wird die gerettete Speicherkonfiguration wieder hergestellt. Diese Routine liegt im Kassettenpuffer. Sollten Sie mit der Datasette arbeiten, legen Sie sie bitte nach $0D00, in den RS232-Eingabepuffer.

Wenn Sie beide Programme eingegeben und gespeichert haben, können Sie mit der Assemblerprogrammierung beginnen. Bis Sie selbst eine komfortablere Lösung gefunden haben, gehen Sie bei der Assemblerprogrammierung bitte wie folgt vor:
1. Basic-Programm laden und starten.
2. Maschinenprogramm laden und mit »SYS DEC("0B00")« aufrufen.

Für Ihre Maschinenprogramme steht Ihnen nun der Bereich $1300 bis $17FF (ab $1800 beginnt der für die Funktionstastenbelegung benötigte Bereich) zur Verfügung. Eventuell können Sie auch weitere Bereiche nutzen, zum Beispiel $0C00 bis $0DFF, wenn Sie die RS232-Schnittstelle, oder den Bereich von $0B00 bis $0BFF, wenn Sie den Kassettenpuffer nicht benötigen.

Die Nutzung aller Bereiche unterhalb $2000 ist theoretisch möglich. Probleme durch das »Absägen des eigenen Astes« werden dank dem bis $1FFF reichenden, gemeinsamen Speicherbereich nicht auftreten.

Um wirklich problemlos in Assembler arbeiten zu können, benötigen Sie zwei weitere Routinen, die zwischen Bank 0 und Bank 1 umschalten. Diese Routinen wurden von mir in das Initialisierungs-Programm integriert.

Initialisierung + Umschaltroutinen:

```
a 00b00  ad  00 ff  lda $ff00
a 00b03  48         pha
a 00b04  a9  00     lda #$00
a 00b06  8d  00 ff  sta $ff00
a 00b09  ad  06 d5  lda $d506
a 00b0c  09  06     ora #$06
a 00b0e  8d  06 d5  sta $d506
a 00b11  68         pla
a 00b12  8d  00 ff  sta $ff00
a 00b15  60         rts
a 00b16  08         php
a 00b17  48         pha
a 00b18  a9  00     lda #$00
a 00b1a  8d  00 ff  sta $ff00
a 00b1d  68         pla
a 00b1e  28         plp
a 00b1f  60         rts
a 00b20  08         php
a 00b21  48         pha
a 00b22  a9  7f     lda #$7f
a 00b24  8d  00 ff  sta $ff00
a 00b27  68         pla
a 00b28  28         plp
a 00b29  60         rts
```

$0B16 ist die Einsprungadresse zum Umschalten auf Bank 0, $0B20 der Einsprung zum Umschalten auf Bank 1 (nur RAM), die wichtigere der beiden Routinen. Wichtiger, da Sie Bank 0 wohl nur zum Lesen von Parametern aus dem Basic-Text verwenden werden, wozu problemlos die entsprechenden Routinen des Basic-Interpreters verwendet werden können.

Assemblerprogrammierung

Im Laufe der Zeit werden sicherlich auch elegantere Lösungen der genannten Probleme entdeckt werden. Dieser Artikel sollte nur zeigen, worauf Sie bei der Assemblerprogrammierung des C 128 achten müssen:
1. Probleme beim Umschalten zwischen den Speicherbänken.
2. Probleme mit Überschneidungen zwischen als gemeinsam definierten Speicherbereichen, zum Beispiel Bildschirm und Variablentabelle.
3. Probleme bei dem Zugriff auf Betriebssystem-Routinen, deren Lösung davon abhängt, in welchen Bereich die Programme gelegt werden, unter das ROM oder in einen reinen RAM-Bereich.

Wie Sie sehen, ist es bei der Programmierung des C 128 in Assembler außerordentlich wichtig, alle Zugriffe, die im späteren Programm sowohl auf verschiedene RAM-Bänke als auch auf ROM-Routinen erfolgen sollen, von vornherein einzuplanen. Ich möchte C 128 Besitzer keinesfalls frustrieren, aber meiner Ansicht nach ist dieser Computer bei der Programmierung in Maschinensprache außerordentlich unkomfortabel und stellt weit höhere Anforderungen an den Programmierer als zum Beispiel der C 64.

Bildschirmausgabe

Zum Abschluß noch ein Rat: Sie werden sich vor weitere Probleme gestellt sehen, wenn Sie Programme, die mit einer bestimmten Art der Bildschirmdarstellung arbeiten, zum Beispiel dem 40-Zeichenmodus, auf einen anderen Ausgabemodus umstellen wollen.

Der Grund ist der unterschiedliche Videocontroller. Für den 40-Zeichenmodus ist der VIC, für den 80-Zeichenmodus der VDC zuständig. Beide Controller arbeiten unterschiedlich. Während der VIC den direkten Zugriff auf das Video-RAM gestattet, ist beim VDC nur ein indirekter Zugriff möglich.

Beim C 64 ist es üblich, daß Assembler-Routinen direkt – unter Umgehung der Betriebssystem-Routinen – in das Video-RAM schreiben beziehungsweise daraus lesen. Bei einer eventuellen Umstellung der Zeichendarstellung sind in solchen Routinen umfangreiche Änderungen nötig.

Ich empfehle Ihnen daher, ausschließlich die Betriebssystem-Routinen - zum Beispiel BSOUT - zu verwenden. Wenn diese standardisierten Schnittstellen verwendet werden, können Sie sicher sein, daß eine spätere Programmumstellung auf eine andere Art der Bildschirmdarstellung nicht einem Neuschreiben des Programms gleichkommt.

Daß mit BSOUT Zeichen auf dem Bildschirm ausgegeben werden können, dürfte allgemein bekannt sein. Woran jedoch nicht jeder Programmierer denkt, ist die Möglichkeit, ohne Umgehung des Betriebssystems auch Zeichen vom Bildschirm zu lesen, indem dieser als logische Datei eröffnet und die Eingabe auf diese Datei gelegt wird.

Zweifellos eine umständliche Methode, die jedoch bei späteren Programmänderungen viel Ärger ersparen kann.

(S. Baloui / ah)

C 128 um 35% schneller

Hier haben wir zwei Leckerbissen für Besitzer des C 128: Ein Programm, das die Benutzung aller C 128-Tasten im C 64-Modus gestattet und ein Programm, das den Commodore 128 im C 64-Modus um 35 Prozent beschleunigt.

Will man einen Commodore 128 im C 64-Modus betreiben, muß man auf vieles verzichten, was im C 128-Modus bereitsteht. Die Zehnertastatur läßt sich nicht benutzen und der FAST-Befehl, der den Commodore 128 um 100 Prozent beschleunigt, ist nur auf dem 80-Zeichen-Monitor anwendbar, der vom C 64 nicht unterstützt wird.

Zwei Programme schaffen hier Abhilfe. Das erste namens »key 128« (Listing 1) gestattet die Benutzung sämtlicher C 128-Tasten im C64-Modus, bis auf die Taste ASCII/DIN, deren Zustand in Speicherzelle 1 steht und die Taste 40/80 DISPLAY, die im C 64-Modus nicht abgefragt werden kann.

»key128« wird als Basic-Lader und nicht als MSE-Listing abgedruckt, was den Vorteil hat, daß es sich an jede beliebige Speicherstelle legen läßt. Nach dem Programmlauf werden SYS-Befehle für Ein- und Ausschalten der C 128 Tastatur angegeben. Die Tasten geben, mit »GET A$« abgefragt, folgende ASCII-Codes zurück:

Taste:	Code:	Shift:
ESC:	chr$(27),	chr$(27)
TAB:	chr$(9),	chr$(24)
ALT:	chr$(14),	chr$(142)
HELP:	chr$(8),	chr$(9)
LINE FEED:	chr$(10),	chr$(10)
NO SCROLL:	chr$(3),	chr$(3)

Mit Hilfe des zweiten Programms »fast64« (Listing 2) läuft ein Commodore 128 im C 64-Modus um 35 Pro-

```
10 REM KEY128 V19-10-85                    <051>
20 REM VON ANDREAS ZELLER, HANAU           <234>
30 :                                       <006>
40 REM AUFGABE: 128ER-TASTEN IM            <196>
50 REM          64ER-MODUS BEDIENEN.       <171>
60 :                                       <036>
70 REM DURCH AENDERN DER ZEILE 170         <203>
80 REM KANN DAS PROGRAMM IN BELIEBIGE      <193>
90 REM BEREICHE GELEGT WERDEN.             <111>
100 :                                      <076>
110 REM WENN DAS PROGRAMM ZUSAMMEN MIT     <247>
120 REM "FAST64" LAUFEN SOLL, MUSS         <156>
130 REM 395 POKE CODE+74,44                <247>
140 REM EINGEFUEGT WERDEN.                 <036>
150 :                                      <126>
160 :                                      <136>
170 CODE=49152:REM STARTADRESSE            <026>
180 :                                      <156>
190 POKE 53265,11:POKE 53296,1             <106>
200 :                                      <176>
210 DIM P(12):FOR J=0 TO 12:READ P:P(J)=CO
    DE+P:NEXT                              <171>
220 :                                      <196>
230 DATA 0,74,110,171,195,213,232,233,234,
    235,236,224,225                        <250>
240 :                                      <216>
250 FOR I=0 TO 284:READ X$                 <138>
260 :                                      <238>
270 REM UMWANDLUNG ADRESSEN                <236>
280 A$=RIGHT$(X$,1):B$=LEFT$(X$,1)         <124>
290 IF A$="+"THEN P=P(ASC(B$)-65):X=P/256:
    GOTO 360                               <072>
300 IF A$="-"THEN P=P(ASC(B$)-65):X=(P/256
    -INT(P/256))*256:GOTO 360              <220>
310 :                                      <032>
320 REM UMWANDLUNG HEX => DEC              <036>
330 :                                      <052>
340 X=ASC(A$)+(A$>"@")*55+(A$<":")*48      <048>
350 X=X+(ASC(B$)+(B$>"@")*55+(B$<":")*48)*
    16                                     <107>
360 POKE CODE+I,X:NEXT                     <255>
370 :                                      <092>
380 POKE 53296,0:POKE 53265,27             <005>
390 :                                      <112>
400 PRINT">DAS PROGRAMM BENUTZT DEN BEREIC
    H"                                     <244>
410 PRINT" VON"CODE"-"CODE+284"."          <012>
420 PRINT                                  <012>
430 PRINT">TABELLE NORMAL AB:"CODE+237";   <217>
440 PRINT"{6SPACE}MIT SHIFT AB:"CODE+261". <043>
450 PRINT                                  <042>
460 PRINT">TASTEN AN MIT: SYS"CODE";"      <185>
470 PRINT"{7SPACE}AUS MIT: SYS"CODE+80"."  <173>
480 :                                      <202>
490 :                                      <212>
500 END                                    <248>
510 :                                      <232>
520 :                                      <244>
530 :                                      <254>
1000 DATA  08,48,78,AD,14,03,C9,C-         <216>
1010 DATA  AD,15,03,E9,C+,F0,0C,AD         <051>
1020 DATA  14,03,8D,G-,G+,AD,15,03         <102>
1030 DATA  8D,H-,H+,AD,02,03,C9,B-         <227>
1040 DATA  AD,03,03,E9,B+,F0,0C,AD         <146>
1050 DATA  02,03,8D,I-,I+,AD,03,03         <222>
1060 DATA  8D,J-,J+,A9,C-,8D,14,03         <200>
1070 DATA  A9,C+,8D,15,03,A9,B-,8D         <057>
1080 DATA  02,03,A9,B+,8D,03,03,68         <004>
1090 DATA  28,60,20,A-,A+,6C,I-,I+         <011>
1100 DATA  08,48,78,AD,G-,G+,8D,14         <189>
1110 DATA  03,AD,H-,H+,8D,15,03,AD         <116>
1120 DATA  I-,I+,8D,02,03,AD,J-,J+         <054>
1130 DATA  8D,03,03,68,28,60,A9,40         <193>
1140 DATA  85,CB,29,00,8D,2F,D0,09         <189>
1150 DATA  FF,8D,00,DC,CD,01,DC,F0         <159>
1160 DATA  42,A0,59,A9,FB,8D,2F,D0         <130>
1170 DATA  48,AD,01,DC,CD,01,DC,D0         <133>
1180 DATA  F8,A2,08,0A,B0,02,84,CB         <005>
1190 DATA  88,C0,41,90,07,CA,D0,F3         <160>
1200 DATA  68,4A,10,E1,68,AD,8D,02         <182>
1210 DATA  0A,C9,08,90,02,A9,06,AA         <101>
1220 DATA  BD,L-,L+,85,F5,BD,M-,M+         <173>
1230 DATA  85,F6,20,F-,F+,20,E0,EA         <048>
1240 DATA  20,F-,F+,A9,FF,8D,2F,D0         <079>
1250 DATA  29,7F,8D,00,DC,A4,CB,8C         <164>
1260 DATA  K-,K+,6C,G-,G+,A5,C5,AE         <112>
1270 DATA  K-,K+,8D,K-,K+,86,C5,60         <114>
1280 DATA  D-,D+,E-,E+,E-,E+,D-,D+         <160>
1290 DATA  31,EA,83,A4,40                  <252>
1300 DATA  08,38,35,09,32,34,37,31         <201>
1310 DATA  1B,2B,2D,0A,0D,36,39,33         <163>
1320 DATA  0E,30,2E,91,11,9D,1D,03         <020>
1330 DATA  09,38,35,18,32,34,37,31         <231>
1340 DATA  1B,2B,2D,0A,0D,36,39,33         <193>
1350 DATA  8E,30,2E,91,11,9D,1D,03         <114>
```


Listing 1. Mit diesem Listing können Sie im C 64-Modus des C 128 alle Tasten abfragen (auch zum Beispiel die Zehnertastatur).




```
10 REM FAST64 V19-10-85                          <209>
20 REM VON ANDREAS ZELLER, HANAU                 <234>
30 :                                             <006>
40 REM AUFGABE: 128ER IM 64ER-MODUS              <144>
50 REM            UM 35% BESCHLEUNIGEN.          <062>
60 :                                             <036>
70 REM DURCH AENDERN DER ZEILE 170               <203>
80 REM KANN DAS PROGRAMM IN BELIEBIGE            <193>
90 REM BEREICHE GELEGT WERDEN.                   <111>
100 :                                            <076>
110 REM WENN DAS PROGRAMM ZUSAMMEN MIT           <247>
120 REM "KEYS128" LAUFEN SOLL, MUSS              <250>
130 REM FOLGENDE ZEILE EINGEFUEGT WERDEN:        <003>
140 REM 395 POKE CODE+110,44                     <038>
150 :                                            <126>
160 :                                            <136>
170 CODE=49440:REM STARTADRESSE                  <012>
180 :                                            <156>
190 POKE 53265,11:POKE 53296,1                   <106>
200 :                                            <176>
210 DIM P(6):FOR J=0 TO 6:READ P:P(J)=CODE
    +P:NEXT                                      <103>
220 :                                            <196>
230 DATA 0,110,159,216,217,218,219               <245>
240 :                                            <216>
250 FOR I=0 TO 219:READ X$                       <141>
260 :                                            <238>
270 REM UMWANDLUNG ADRESSEN                      <236>
280 A$=RIGHT$(X$,1):B$=LEFT$(X$,1)               <124>
290 IF A$="+"THEN P=P(ASC(B$)-65):X=P/256:
    GOTO 360                                     <072>
300 IF A$="-"THEN P=P(ASC(B$)-65):X=(P/256
    -INT(P/256))*256:GOTO 360                    <220>
310 :                                            <032>
320 REM UMWANDLUNG HEX => DEC                    <036>
330 :                                            <052>
340 X=ASC(A$)+(A$>"@")*55+(A$<":")*48            <048>
350 X=X+(ASC(B$)+(B$>"@")*55+(B$<":")*48)*
    16                                           <107>
360 POKE CODE+I,X:NEXT                           <255>
370 :                                            <092>
380 POKE 53296,0:POKE 53265,27                   <005>
390 :                                            <112>
400 PRINT">DAS PROGRAMM BENUTZT DEN BEREIC
    H"                                           <244>
410 PRINT" VON"CODE"-"CODE+219"."                <108>
420 PRINT                                        <012>
430 PRINT">SCHNELL MIT: SYS"CODE";"              <085>
440 PRINT"{2SPACE}NORMAL MIT: SYS"CODE+116
    "."                                          <160>
450 :                                            <172>
460 END                                          <208>
470 :                                            <192>
480 :                                            <202>
490 :                                            <212>
1000 DATA  08,48,78,AD,14,03,C9,C-               <216>
1010 DATA  AD,15,03,E9,C+,F0,0C,AD               <051>
1020 DATA  14,03,8D,D-,D+,AD,15,03               <181>
1030 DATA  8D,E-,E+,AD,02,03,C9,B-               <028>
1040 DATA  AD,03,03,E9,B+,F0,0C,AD               <146>
1050 DATA  02,03,8D,F-,F+,AD,03,03               <045>
1060 DATA  8D,G-,G+,A9,C-,8D,14,03               <001>
1070 DATA  A9,C+,8D,15,03,A9,B-,8D               <057>
1080 DATA  02,03,A9,B+,8D,03,03,A9               <017>
1090 DATA  00,8D,30,D0,AD,12,D0,D0               <058>
1100 DATA  FB,A9,31,8D,12,D0,AD,11               <078>
1110 DATA  D0,29,7F,8D,11,D0,A9,01               <251>
1120 DATA  8D,30,D0,AD,1A,D0,09,01               <225>
1130 DATA  8D,1A,D0,68,28,60,20,A-               <007>
1140 DATA  A+,6C,F-,F+,08,48,78,AD               <104>
1150 DATA  1A,D0,29,FE,8D,1A,D0,A9               <112>
1160 DATA  00,8D,30,D0,AD,D-,D+,8D               <127>
1170 DATA  14,03,AD,E-,E+,8D,15,03               <202>
1180 DATA  AD,F-,F+,8D,02,03,AD,G-               <052>
1190 DATA  G+,8D,03,03,68,28,60,AC               <047>
1200 DATA  19,D0,30,07,AD,0D,DC,58               <042>
1210 DATA  6C,D-,D+,98,4A,90,F9,8C               <011>
1220 DATA  19,D0,AD,30,D0,49,01,8D               <053>
1230 DATA  30,D0,4A,B0,03,A9,FA,2C               <118>
1240 DATA  A9,31,8D,12,D0,68,A8,BA               <251>
1250 DATA  BD,04,01,D0,03,DE,05,01               <154>
1260 DATA  DE,04,01,68,AA,68,28,60               <221>
1270 DATA  31,EA,83,A4                           <145>
```

Listing 2. Mit »fast 64« läuft der C128 im C64-Modus um 35 Prozent schneller.

zent schneller. Dies geschieht mit Hilfe der Taktumschaltung von 1 MHz auf 2 MHz. Normalerweise erscheint bei der Umschaltung auf 2 MHz »Speichermüll« auf dem Bildschirm, da der Video-Chip nicht mehr auf den Speicher zugreifen kann. Deswegen wird beim Commodore 128 beim FAST-Befehl der 40-Zeichen-Videochip auch einfach abgeschaltet. Unser Programm umgeht diese Einschränkung so, daß der 2 MHz-Modus nur eingeschaltet wird, wenn der Videochip den Rahmen oben und unten aufbaut und ohnehin nicht auf den Speicher zugreift.

Während die Beschleunigung wirksam ist, können Peripheriegeräte nicht korrekt benutzt werden (das hängt mit der IRQ-Routine zusammen). Vor Laden/Speichern auf Diskette oder Cassette sowie Drucken sollte das Programm deshalb wieder abgeschaltet werden (Listing 2, Zeile 440)!

Auch dieses Programm wird als Basic-Lader veröffentlicht und läßt sich somit in jeden beliebigen Speicherbereich legen. Beide Programme sind gegen RUN/STOP-RESTORE geschützt, das heißt, daß sie auch nach dem Drücken dieser Tastenkombination noch aktiviert sind. Sie laufen über den IRQ-Vektor. Wenn beide Programme gleichzeitig benutzt werden, sollten Sie die im Listing in den REM-Zeilen angegebenen Änderungen vornehmen. Diese Zeilen müssen Sie natürlich nicht mit abtippen. »fast 64« muß zuletzt aktiviert werden.

Wenn Sie diese Hilfen in Ihren eigenen Programmen benutzen wollen, sollte es keinerlei Anpassungsprobleme geben; es sei denn, Sie arbeiten mit Interrupt-Steuerung. In diesem Fall sollten Sie die Programme erst aktivieren, nachdem Ihr Interrupt eingerichtet ist. Dasselbe gilt natürlich auch für Spiele und kommerzielle Programme. Hier wird man in den meisten Fällen nicht umhin kommen, das jeweilige Programm mit Hilfe eines Monitors auf Veränderungen des IRQ-Vektors zu untersuchen.

Noch ein Hinweis: Beide Programme verwenden indirekte Sprungbefehle, bei denen der 6510/8510-Prozessor unter gewissen Umständen (beim Verwenden von kritischen Adressen, zum Beispiel JMP($20FF) oder JMP($21FF) etc.) Fehler macht. Der 6502-Befehl JMP (addr) funktioniert nämlich nur so lange korrekt, wie das Lo-Byte von »addr« ungleich $FF ist. Ansonsten wird die Adresse nicht aus »addr/addr + 1« geholt, sondern aus »addr/addr–$FF«!
Beispiel: JMP ($0103) springt zur Adresse, die in ($0103/$0104) steht, aber JMP ($01FF) springt zur Adresse, die in ($01FF/$0100) steht! Sollte ein Programm nicht laufen, erhöhen Sie die Startadresse um 1. Die im Listing angegebenen Startadressen sind selbstverständlich geprüft.

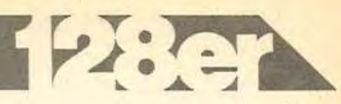
Anmerkung: Die Beschleunigung um 35 Prozent läßt sich auch im C 128-Modus programmieren. Wann erscheint das erste Listing dazu? Profis mögen uns schreiben!

(Andreas Zeller/tr)

Rätselfreunde aufgepaßt!

85 72 96 6 89 59 32 121
74 105 10 125
5 88 26 112 51 68
49 102 38 91
29 3 75 18 2 19 40
14 127 52
84
134 46
4 114
1
34 79 86
71 103
17 80 45
9 101 129
76 83 70 87 22 43 37
62 20
131 117 110
30 47
12 66 57
11 27 108 58
78 132 111
92 69 63
64
16 135 73 7 39
90 124 13 24
100 97 119 61
60 21 82 93 77
81 31 123 128 35 104
28 115 113 25
23 8 67
106 94 130 65
50 48 120
95 55 56
99 36 33 98
126 109 107 116
53 122 15
133
42 118 54
41 44

Star-Drucker zu gewinnen! Finden Sie das Lösungswort und gewinnen Sie einen von 50 tollen Preisen.



Nein, hier finden Sie nicht etwa ein neues Abenteuer-Spiel zum Abtippen, sondern ein »echtes« Kreuzworträtsel. Und bei den tollen Preisen, die es dabei zu gewinnen gibt, lohnt sich das Mitmachen wirklich!

Bevor Sie sich jedoch in unser Super-Rätsel stürzen, einige kleine »Bedienungshinweise«:

1. Es dürfen nur Buchstaben eingesetzt werden. Falls aber doch Zahlen im Lösungswort vorkommen sollten, so sind diese auszuschreiben (zum Beispiel »dreizehn« etc.).

2. Die Umlaute ä, ö und ü sind mit zwei Buchstaben zu schreiben, also ae, oe und ue.

3. Die etwas dickeren Trennstriche zwischen den einzelnen Feldern stehen als Begrenzungen zwischen den gesuchten Wörtern.

4. Wenn eine Zahl im oberen Teil eines Kästchens steht, so bedeutet dies, daß das Lösungswort ab hier SENKRECHT einzutragen ist. Entsprechend bedeutet eine im unteren Teil stehende Zahl, daß eine WAAGERECHTE Eintragung erwartet wird.

5. Die elf farbig gekennzeichneten Felder sind die einzelnen Buchstaben des Lösungswortes. Diese ergeben, in die richtige Reihenfolge gebracht, das gesuchte Lösungswort.

6. Die sieben dunkelgrauen Rechtecke dürfen nicht ausgefüllt werden.

Wir haben die Fragen absichtlich nicht leicht gewählt, denn was nützt das tollste Rätsel, wenn man nach zehn Minuten Knobeln schon die Lösung hat. Wenn Sie die Artikel in diesem Heft aufmerksam lesen, werden Sie sicher einen Großteil der Fragen beantworten können. Zum Schluß noch eine kleine Bitte: Auf der Postkarte mit dem Lösungswort beantworten Sie bitte noch die folgenden drei Fragen:

1. Welcher Artikel in diesem Sonderheft hat Ihnen am besten gefallen?
2. Welchen Computer/Floppy besitzen Sie ?
3. Wie alt sind Sie?

Und damit Ihre Mühe nicht ganz umsonst war, gibt es natürlich ein paar tolle Preise zu gewinnen. (tr)

Übrigens:
Das komplette Kreuzworträtsel wurde von unserem Programm aus der 64'er, Ausgabe 12/84, erstellt.

1. Preis: Der Super-Drucker von Star, der SG10 (Test in der 64'er, Ausgabe 5/85)

2. Preis: Die drei CP/M-Profi-Programme für den Commodore 128: WordStar, dBase II und Multiplan als Paket

3. Preis: Eines der oben genannten Programme zur freien Auswahl

4. bis 50. Preis Ein Buchgutschein im Wert von 50 Mark vom Markt & Technik Verlag.

Die Postkarte schicken Sie bitte an:

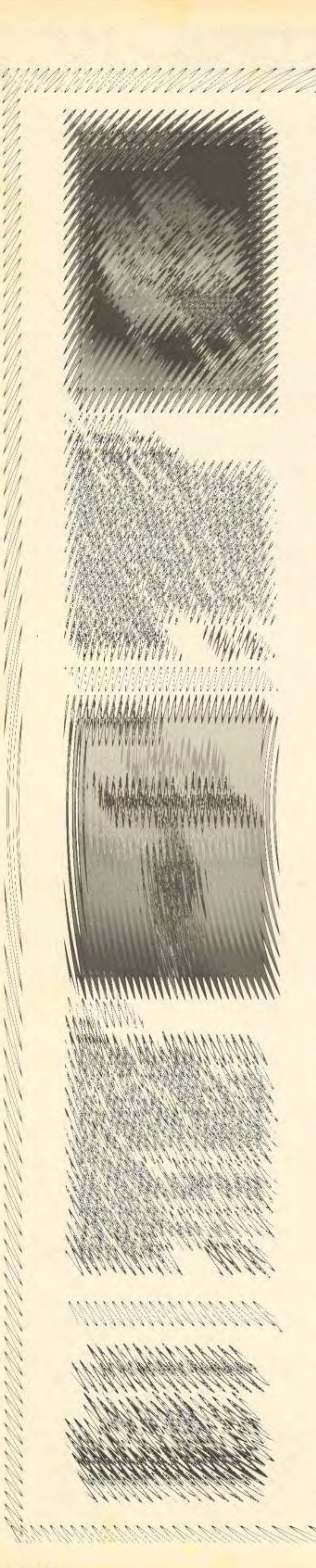
Markt & Technik Verlag
Aktiengesellschaft
Redaktion 64'er
»Kreuzworträtsel«
Hans-Pinsel-Straße 2
8013 Haar bei München
Einsendeschluß: 15. 1. 1986

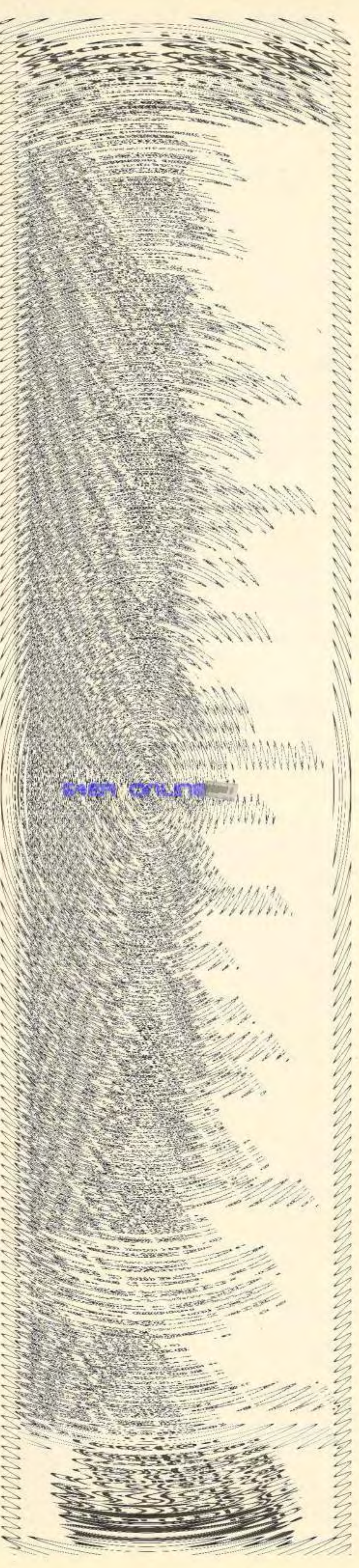
Senkrecht

(2) Wieviele Prozessoren hat der C 128? (3) Abkürzung für Mikroprozessor; (4) Zahlensystem; (5) Büromaschinenhersteller; (6) englisch: Zeichenkette; (12) bringt Daten schwarz auf weiß; (13) Textverarbeitung für CP/M; (17) neuer Super-Computer von Commodore; (18) Soundchip des C 128/C 64; (19) Wieviele Farben hat der C 64/C 128? (21) internes Zahlensystem des Computers; (22) unser Maschinenmonitor; (24) Speichermedium einer Floppy; (25) musikalischer Basic-Befehl des C 128; (26) größter Feind der DFÜ-Fans; (28) Programmiersprache; (29) logischer Operator; (32) wird zur DFÜ benötigt; (34) Basic-Befehl: Variablenzuweisung; (37) Neustart des Computers; (40) billigstes Speichergerät zum C 64; (43) brennbarer Computerbaustein; (45) Monitornorm; (51) Datenübertragung zur Floppy; (55) programmierbare Logikschaltung; (56) überflüssiger Basic-Befehl; (57) Nichts-tu-Befehl; (59) Gegenteil von out; (60) untrennbar mit NEXT verbunden; (64) Eingabehilfe; (65) damit wird dBase verlassen; (66) Prozessorregister; (68) Gerät zur Computerverbindung per Telefon; (70) Jargon für Diskettenstation; (72) englisch für »Stapel«; (75) billiger Massenspeicher; (76) Logische Speichereinheit; (77) Meldung über einen Gerätezustand; (78) Programm zur Texteingabe/Korrektur; (79) Rückkehr in den Anfangszustand; (80) Abkürzung für elektronische Datenverarbeitung; (82) Abkürzung für »Megahertz«; (83) Abkürzung für »Hertz«; (84) ASCII-Zeichen für Zeilenvorschub; (85) Logische »Nicht-Und«-Verknüpfung; (86) Nur-Lese-Speicher; (87) Abkürzung für »Zeilendrucker«; (88) Basic-Befehl für »Integer«; (89) lese- und beschreibbarer Speicher; (90) Abkürzung für »Digital nach Analog«; (93) maximale Adresse für 8-Bit-Computer (Hex); (94) Abkürzung für »höherwertiges Bit«; (95) Mikroschalter auf der Platine; (96) Anweisung für das Ende eines Programmes; (106) Abkürzung für »Computer-gestützter Entwurf«; (107) Abkürzung für »Exclusiv-Oder«-Verknüpfung; (109) Technik zur Herstellung von ICs; (110) Kurzbezeichnung für »eine Binärziffer«; (111) Milliampere; (112) Per annum; (113) Masseneinheit für Luftdruck; (114) Kilo-Ampere; (115) Kurzbezeichnung für »Technische Universität«; (116) Englische Abkürzung für »Pfund«; (117) Abkürzung für ein englisches Längenmaß; (121) Corps Consulaire; (124) Holländischer Gulden; (126) Abkürzung für »Jahrhundert«; (130) Sommersemester; (131) Operationssaal; (132) Evangelisch; (133) Abkürzung für eine Lichteinheit;

Waagerecht

(1) Zeitschriftenverlag; (7) Videochip des C 128/C 64; (8) Übersetzer für Programmiersprache; (9) Düsseldorfer Software-Hersteller; (10) bewegliches Grafikobjekt; (11) letzter Befehl eines Basic-Programms; (14) Schreib-/Lesespeicher; (15) Zusatzgerät zu jedem Computer; (16) Abkürzung für Datenfernübertragung; (20) Übersetzer für Maschinensprache; (23) Wichtige Funktionstaste des C 128; (27) uraltes Freß-Spiel; (30) Hersteller des Z80-Prozessors; (31) Verbindung zwischen C 64 und Floppy; (33) Sitz der 64'er Redaktion; (35) Nur-Lese-Speicher; (36) Starten eines Maschinenprogramms; (38) Basic-Befehl zur Ausgabe von Zeichen; (39) Basic-Eingabe-Befehl; (41) Datensichtgerät; (42) genialer Floppybeschleuniger; (44) ROM des Betriebssystems; (46) halbes Byte; (47) kleinstes Teil eines Byte; (48) tödlich für jedes Basic-Programm; (49) Port-Baustein; (50) modernes Eingabegerät; (52) Schleifenanfang; (53) ohne das läuft WordStar nicht; (54) Befehl zum Laden; (58) Deutsche Industrie-Norm; (61) Datenbanksprache; (62) Digitalpegel; (63) Divisionsrest; (67) diese Taste gibt es dreimal auf dem C 64/C 128; (69) Basic-Befehl zum Laden des Speichers; (71) Höhere kommerzielle Programmier-Sprache; (73) Basic-Befehl zum Lesen des Speichers; (74) Bildschirmausschnitt; (81) Abkürzung für »Television«; (91) Steuerzeichen für Kontrollcodes; (97) Übertragungsrate von Daten; (98) Internationale Normierungsorganisation; (99) Abkürzung für »Analog nach Digital«; (100) Binäre Darstellung von dezimalen Ziffern; (101) Legendärer Commodore-Computer; (102) Abkürzung für »Fernschreiber«; (103) Abkürzung für »Zeichen löschen«; (104) Signal zur Anzeige eines Interrupts; (105) Anweisungen von Programmiersprachen; (108) Ende der Übertragung; (118) Junior; (119) Milligramm; (120) Kilogramm; (122) Kurzbezeichnung für »Lichtjahr«; (123) Firma; (125) Corps Diplomatique; (127) Hektoliter; (128) Abkürzung für »Oberbürgermeister«; (129) Seemeile; (134) Englisch für »per Adresse«; (135) Englisch für »nachmittags«;





Impressum

Herausgeber: Carl-Franz von Quadt, Otmar Weber

Chefredakteur: Michael Scharfenberger

Leitender Redakteur: Albert Absmeier

Redaktion: Volker Everts, Christine Geißler, Achim Hübner, Georg Klinge (Koordination), Harald Meyer, Markus Ohnesorg, Thomas Röder, Boris Schneider, Arnd Wängler

Fremdautoren:
Ponnath, Binz, Reetz, Köhler, Baloui, Hierlmeier, Zeller, Zumbach

Layout:
Leo Eder (Ltg.), Sigrid Kowalewski (Cheflayouterin)

Herstellung: Klaus Buck

Auslandsrepräsentation:

Schweiz: Markt & Technik Vertriebs AG, Kollerstr. 3, CH-6300 Zug, Tel. 042-41 5656, Telex: 862329

USA: M&T Publishing Inc.; 2464 Embarcadero Way, Palo Alto, CA 94303

Manuskripteinsendungen: Manuskripte und Programmlistings werden gerne von der Redaktion angenommen. Sie müssen frei sein von Rechten Dritter. Sollten sie auch an anderer Stelle zur Veröffentlichung oder gewerblichen Nutzung angeboten werden, so muß dies angegeben werden. Mit der Einsendung von Manuskripten und Listings gibt der Verfasser die Zustimmung zum Abdruck in von der Markt & Technik Verlag AG herausgegebenen Publikationen und zur Vervielfältigung der Programmlistings auf Datenträger. Mit der Einsendung von Bauanleitungen gibt der Einsender die Zustimmung zum Abdruck in von Markt & Technik Verlag AG verlegten Publikationen und dazu, daß Markt & Technik Verlag AG Geräte und Bauteile nach der Bauanleitung herstellen läßt und vertreibt oder durch Dritte vertreiben läßt. Honorare nach Vereinbarung. Für unverlangt eingesandte Manuskripte und Listings wird keine Haftung übernommen.

Leitung Marketing Vertrieb: Hans Hörl (114)

Vertriebsleitung: Helmut Grünfeldt (189)

Anzeigenverwaltung und Disposition: Michaela Hörl

Verlagsleiter M&T-Buchverlag: Günther Frank

Druck: Erwin Jungfer GmbH & Co. KG, Am Schloßbahnhof, 3420 Herzberg

Preis: Das Einzelheft kostet DM 14,–

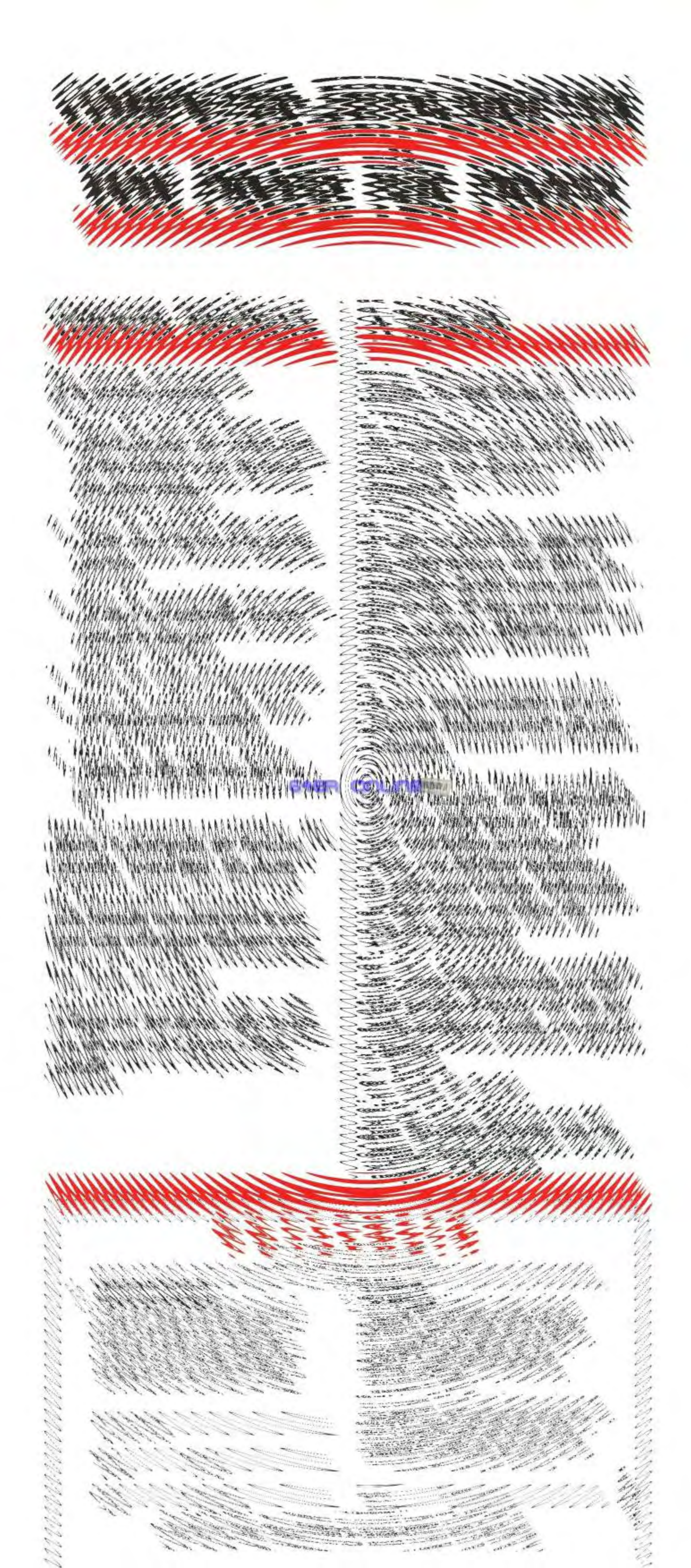
Vertrieb Handelsauflage: Inland (Groß-, Einzel- und Bahnhofsbuchhandel) sowie Österreich und Schweiz: Pegasus Buch- und Zeitschriften-Vertriebs GmbH, Hauptstätter Straße 96, 7000 Stuttgart 1, Telefon (0711) 648330

Verantwortlich:
Für redaktionellen Teil: Michael Scharfenberger
Für Anzeigen: Brigitte Fiebig

Redaktions-Direktor: Michael M. Pauly

Vorstand: Carl-Franz von Quadt, Otmar Weber

Anschrift für Verlag, Redaktion, Vertrieb, Anzeigenverwaltung und alle Verantwortlichen:
Markt & Technik Verlag Aktiengesellschaft, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon (089) 4613-0, Telex 5-22052


64ER ONLINE


Bücher









